Distributed Systems

Chun-Feng Liao

廖峻鋒

Department of Computer Science

National Chengchi University

Distributed Systems

Indirect Communication (Messaging)

Chun-Feng Liao

廖峻鋒

Dept. of Computer Science

National Chengchi University

近期Deadline提醒

- 期中事務
  - 5/8 期中報告due
  - 5/15 現場demo與期中報告口試 (1-4)
  - 5/22 現場demo與期中報告口試 (5-8)
  - 5/29 現場demo與期中報告口試 (9-12)
  - 6/12 期末demo
  - 6/19 期末demo原始碼/slide/影片 上傳期限

Course Evaluation

- Participation 10%
- In-course individual/group assignment 25%
- 期中考 25%
- 學期專案 40%
  - 期中書面15%:分散式技術解析與試用
    - 書面技術解析報告、現場試用demo
  - 期末口頭25%: 分散式技術應用開發
    - 現場報告、系統demo

期中報告: 分散式技術解析與試用

- 目的
  - 善用學期間學得的分散式系統知識，了解最新業界進展
- 條件:
  - 建議專案 (如列表)
  - 自行選定: 要符合分散式系統定義的相關技術
    - 一定要是分散式系統且專案須有一定規模(程式碼2000行以上)
    - 經老師同意才可選擇
    - 來源
      - CNCF 專案 (https://www.cncf.io/)
      - Apache 專案 (https://www.apache.org/index.html#projects-list)
      - 其它在Github中的分散式專案
- 期末展示要包含所選定的專案技術

期中報告: 分散式技術解析

- **篇幅: A4 5頁以上**
  - Introduction
    - Main function of the project, what is the problem it tries to solve?
    - The overall approach taken by the project (core idea)
    - **該分散式系統/技術的應用場景 (Example)，用來說明:**
      - **在什麼場何用來解決什麼問題? 沒有它又如何?**
  - **架構解析**
    - Explain how this technology works
      - **靜態與動態結構解析 (請使用UML Class/Component/Sequence/Activity diagrams)配合文字說明**
      - **此設計如何在某個背景下解決設計問題(如何達成目的)**
  - **實驗紀錄與心得**
    - How to install, configure and use? (**安裝並使用該技術實作一個簡單的範例的過程記錄**)
    - Demo: A hello world example for this technology
  - Evaluation and comments (**至少0.5頁，心得與建議**)
    - **和相近技術的比較**: Why this work is better than others? In which respect?
    - Does the proposed approach solves the problem? How and how well?
    - What can be enhanced?

特別注意: 若使用ChatGPT輔助，驗證正確性為使用者的重要責任，若經查核正確性有問題，或引用錯誤，會嚴重扣分!

建議專案

- Apache Kafka https://kafka.apache.org/
- Apache Pulsar https://pulsar.apache.org/
- Aedes https://github.com/moscajs/aedes
- Apache Camel https://camel.apache.org/
- Apache Ignite https://ignite.apache.org/
- Apache Mina https://mina.apache.org/
- Apache Vysper https://mina.apache.org/vysper-project/
- Apache Zookeeper https://zookeeper.apache.org/
- Apache OpenMeetings https://openmeetings.apache.org/
- Apache Helix https://helix.apache.org/
- Apache Spark https://spark.apache.org/
- Apache TomEE https://tomee.apache.org/

建議專案

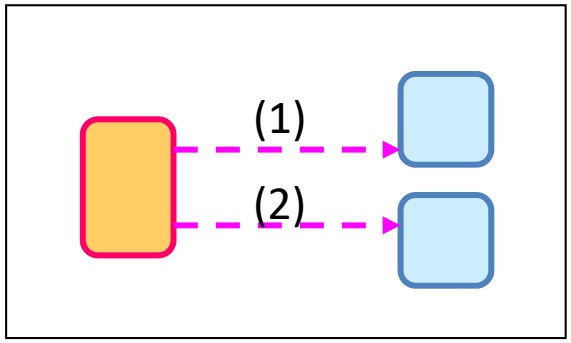
- Operator Framework https://www.cncf.io/projects/operator-framework/
- TiDB https://github.com/pingcap/tidb
- Event Store https://www.eventstore.com/eventstoredb
- CoreDNS https://coredns.io/
- Etcd https://etcd.io/
- OpenTelemetry https://opentelemetry.io/
- Envoy https://www.envoyproxy.io/
- Prometheus https://prometheus.io/

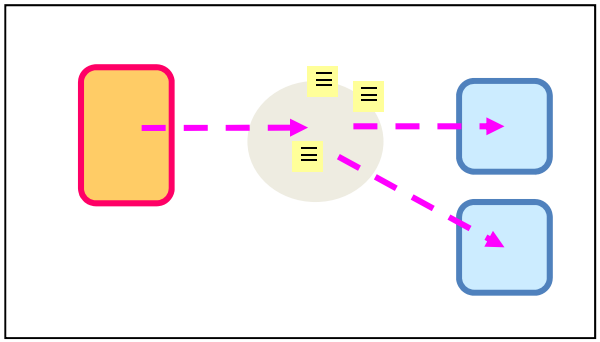
期末Demo

- 目的
  - 就期中小組報告的技術，開發一個實際應用
- 限制
  - 要能實際run
  - 要應用到期中報告的技術
  - 至少3個節點
    - 可以同一台機器，三個Processes，但只透過網路溝通
- 要求
  - 非功能性: 實作至少一項維運功能 (系統偵測、自動回復、隨需伸縮…)
  - 功能性: 要具有意義的應用 (不能是hello word)
- 如何繳交 (評分依據)　未含影片與原始碼Github網址者不予計分!
  - 口頭報告與demo
    - 投影片要上傳到moodle (pdf); demo以影片錄製，上傳到雲端，連結附在投影片上)
  - 原始碼請放到小組其中一員的Github公開repository; 在readme.md中詳述安裝與操作步驟
    - Github網址附在投影片中

Basic Remoting Styles

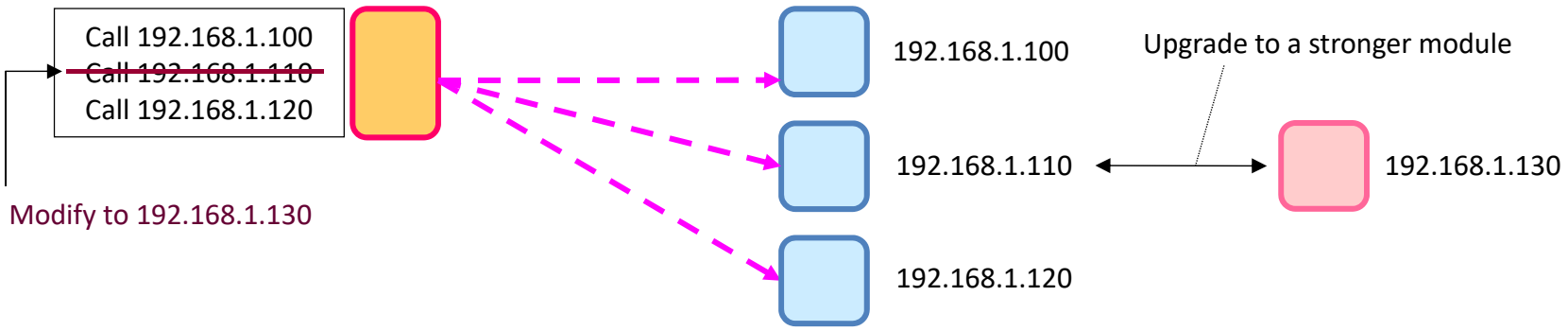
- Two major styles
  - Direct communication (RPC)
  - Indirect communication (Messaging)





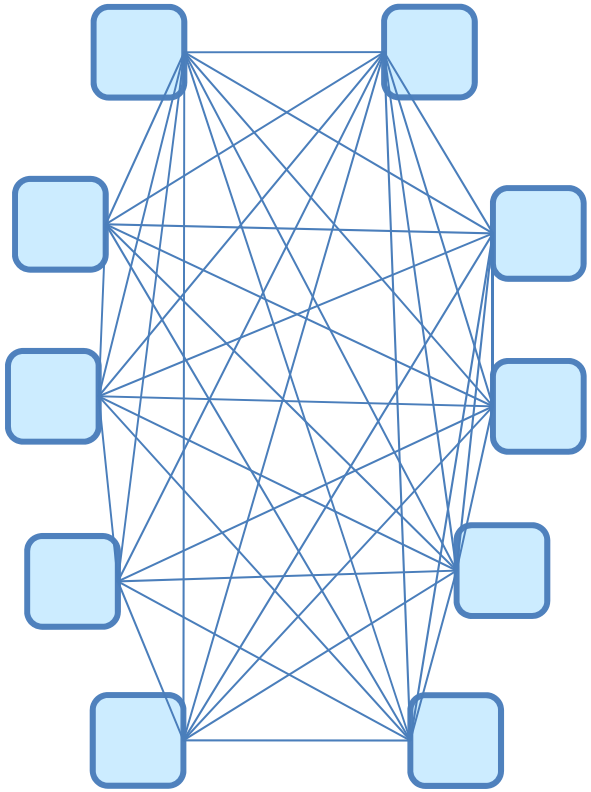
Direct Communication

- Benefits
  - Simplified application development
- Drawbacks
  - Tightly coupled on space (reference) and time (synchronous)
  - Fragile and hard to recover



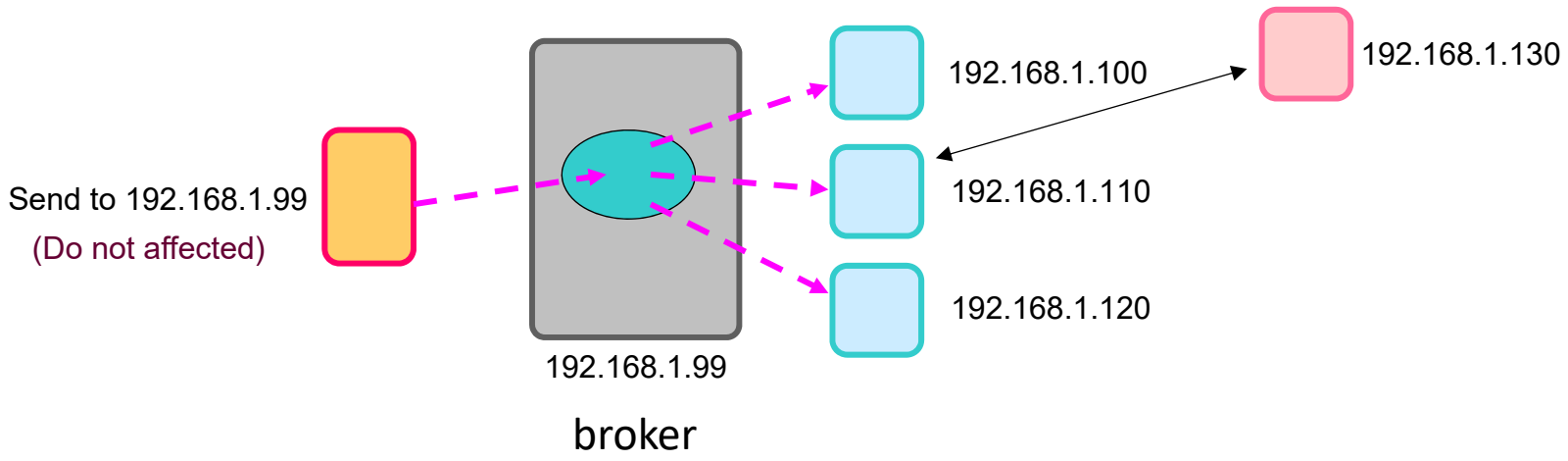
Integrating Systems One-by-One

(n-1)!

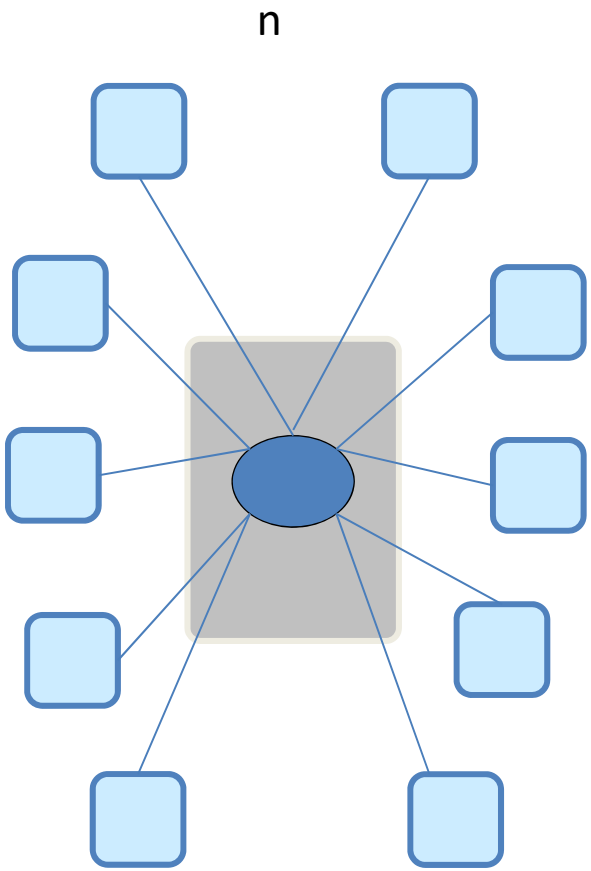


Indirect Communication

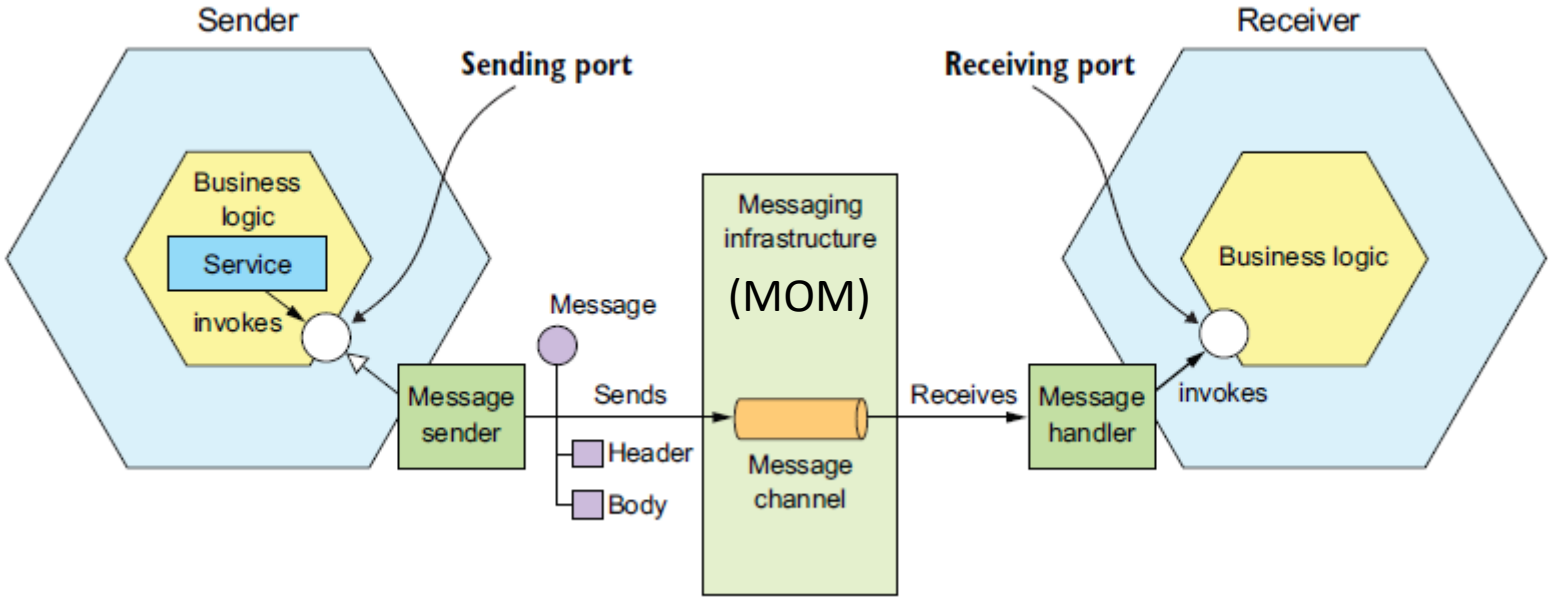
- Benefits
  - Decoupling in both space and time
- Drawbacks
  - Single point of failure can be alleviated by clustering
  - Address binding of the broker can be alleviated by broker discovery
    亦可使用brokerless架構處理(後面會介紹)



Integrating Systems by Using MOM



Architectural Overview



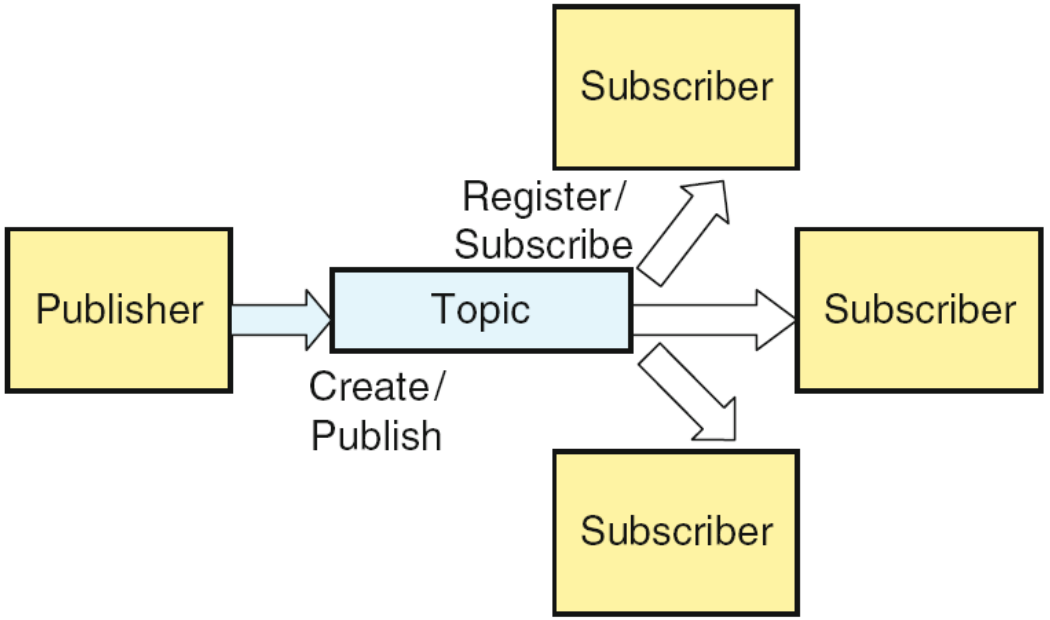
publisher (message) broker subscriber

Message Semantics

- Header
  - Name-value pairs to annotate the message
  - Ex: Metadata, message id, replying channel name
- Body (Payload)
  - Document
    - Ex: return data
  - Command
    - Ex: to simulate RPC
  - Event
    - Indicating that something notable has occurred at a specific time
    - The state change of a domain object at a specific time

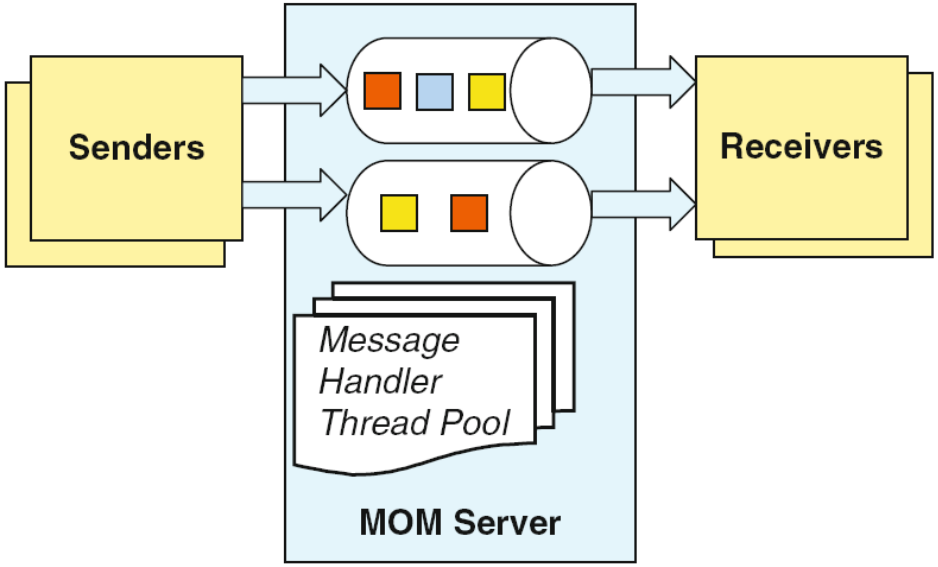
Publish-Subscribe

- Publishers
  - Send a message to a named topic
- Subscribers
  - listen for messages that are sent to topics that interest them



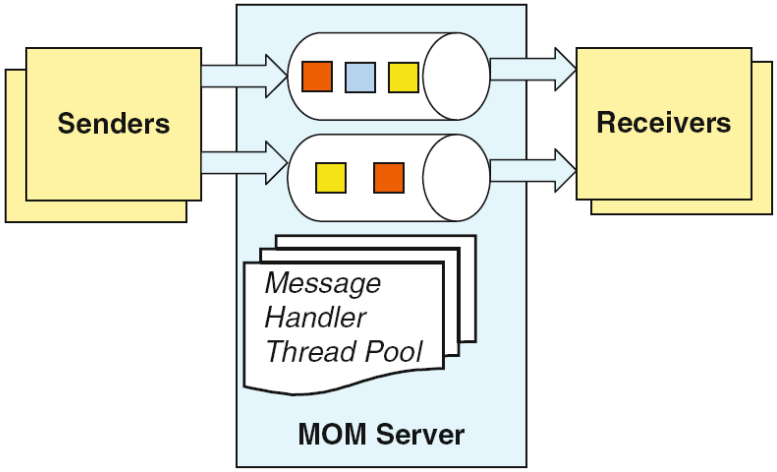
MOM Server (Message Brokers)

- Mechanisms
  - Create and manage multiple messages queues (topics)
  - Handle multiple messages being sent from queues simultaneously using threads organized in a thread pool

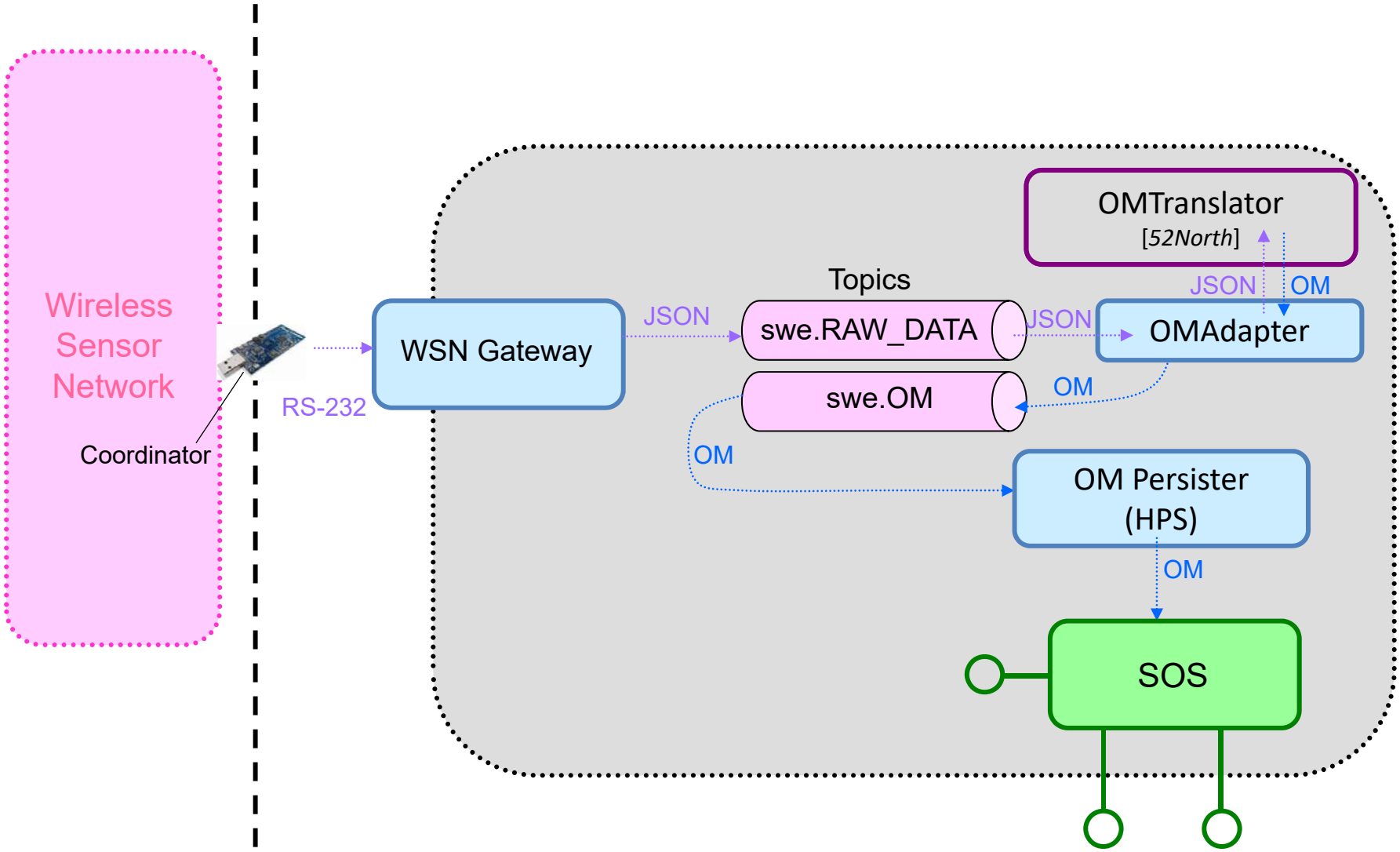


MOM Server (Message Brokers)

- Message transmitting
  - Accept/Ack messages from the senders
  - Place the messages at the end of the queue (topic)
  - Hold messages for an extended period of time



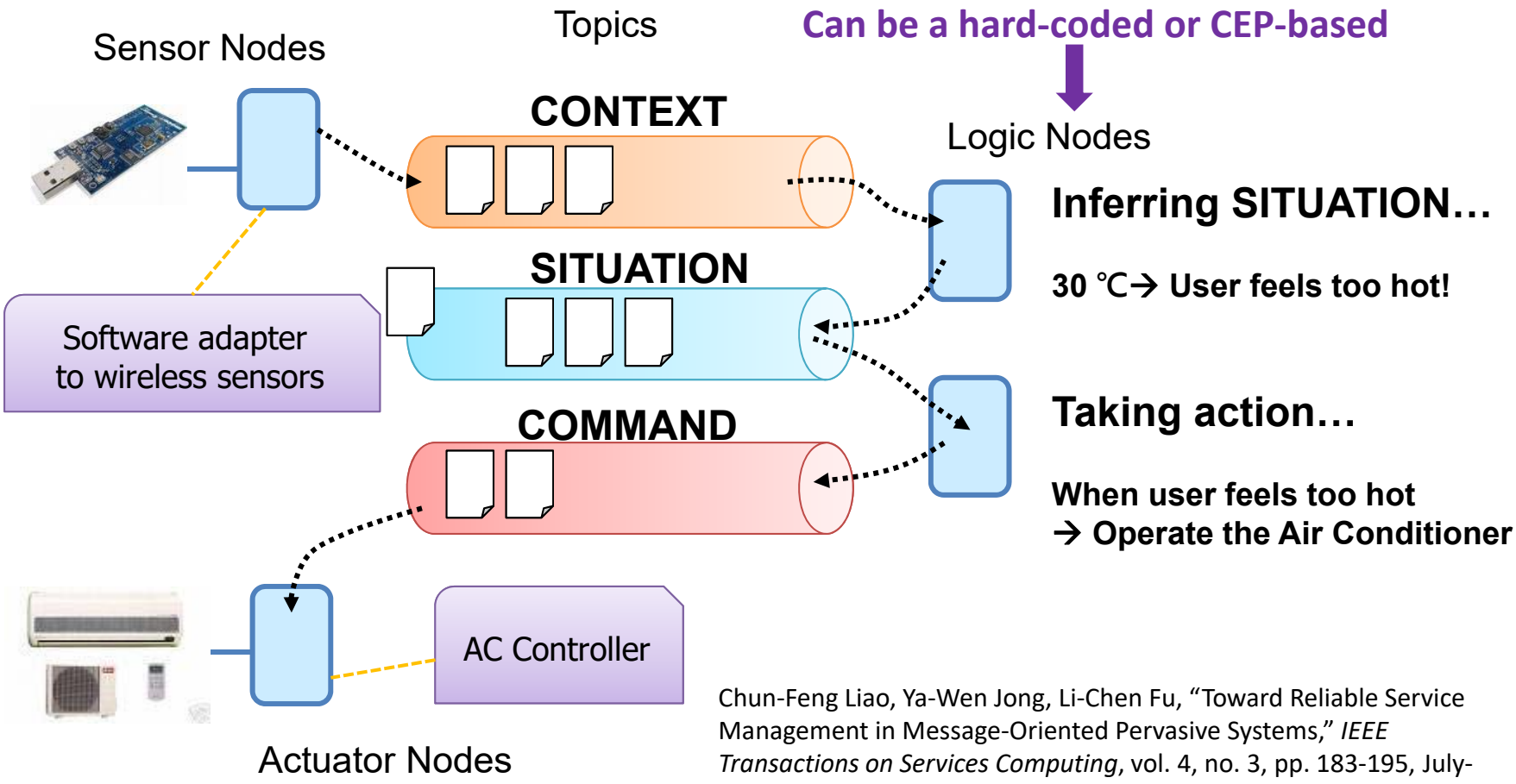
Case:
Sensor Observation Service (OGC SWE)



Case: MQTT

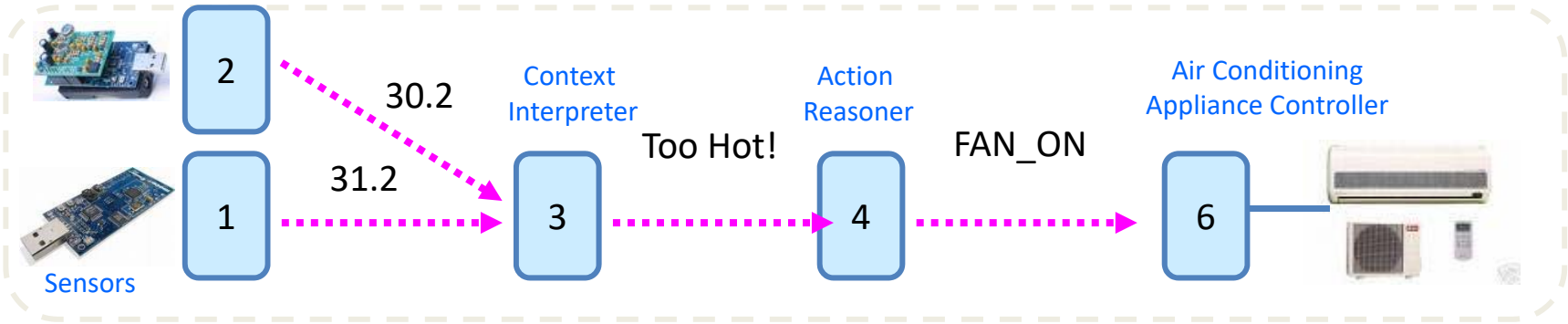
- Message Queuing Telemetry Transport
  - Proposed by IBM and Eurotech
  - Based on TCP/IP (MQTT-SN can use UDP)
- Compact
  - Designed for IoT
    - The most popular MOM for IoT
    - Low bandwidth and computing capability requirement
- Know uses
  - Facebook Messenger, Amazon IoT, OGC, Azure IoT Hub

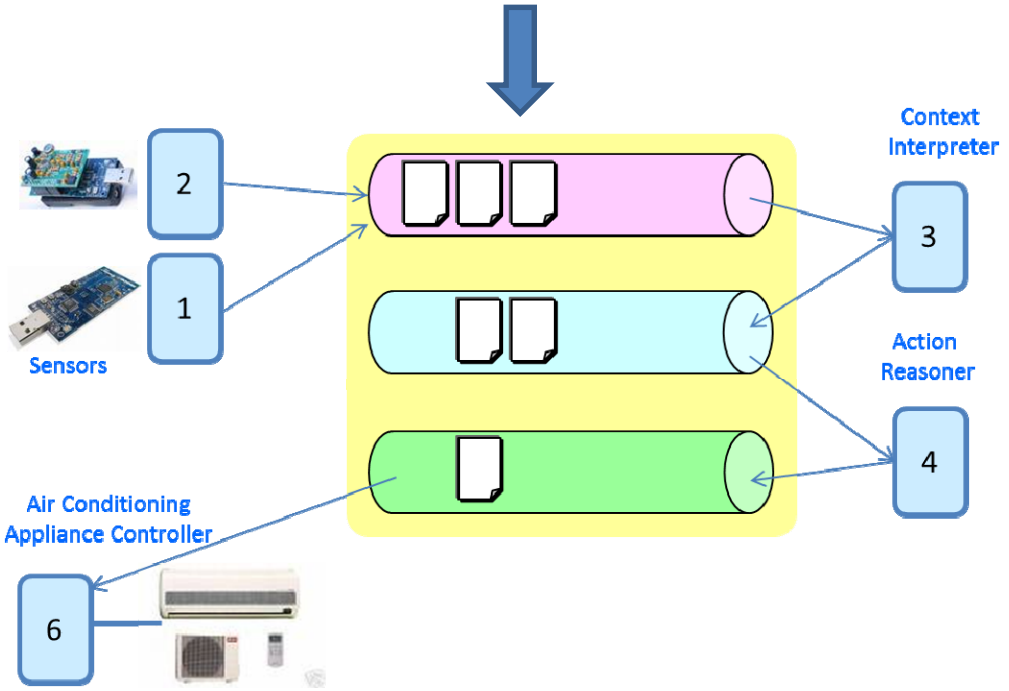
Case: 物聯網系統整合



Chun-Feng Liao, Ya-Wen Jong, Li-Chen Fu, "Toward Reliable Service Management in Message-Oriented Pervasive Systems," *IEEE Transactions on Services Computing*, vol. 4, no. 3, pp. 183-195, July-Sept. 2011.

Case: MQTT 物聯網系統整合

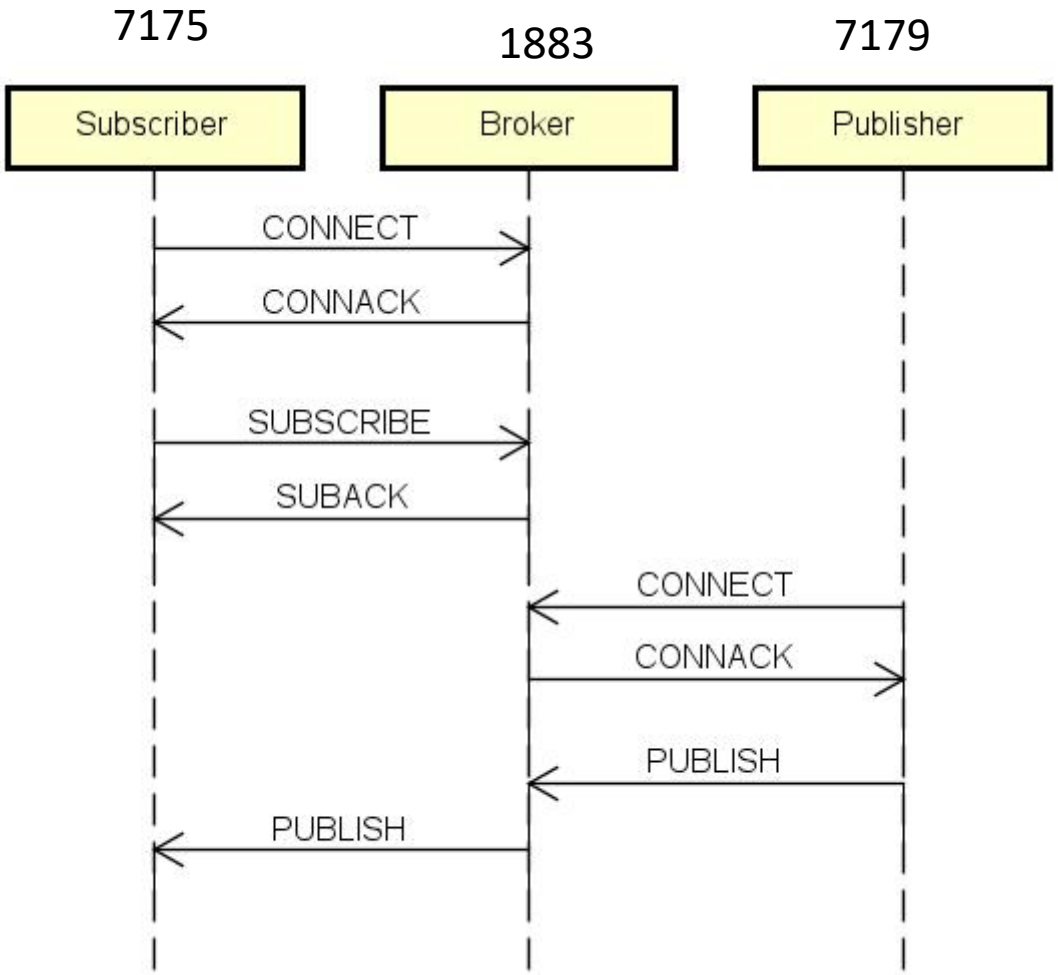




MQTT Basic Protocol Elements

- 連線管理
  - CONNECT/CONNACK
  - DISCONNECT
- 訂閱
  - SUBSCRIBE/ SUBACK
- 發送
  - QoS0: PUBLISH
  - QoS1: PUBACK
  - QoS2: PUBREC/PUBREL/PUBCOMP

訊息傳送基本流程 (QoS0)

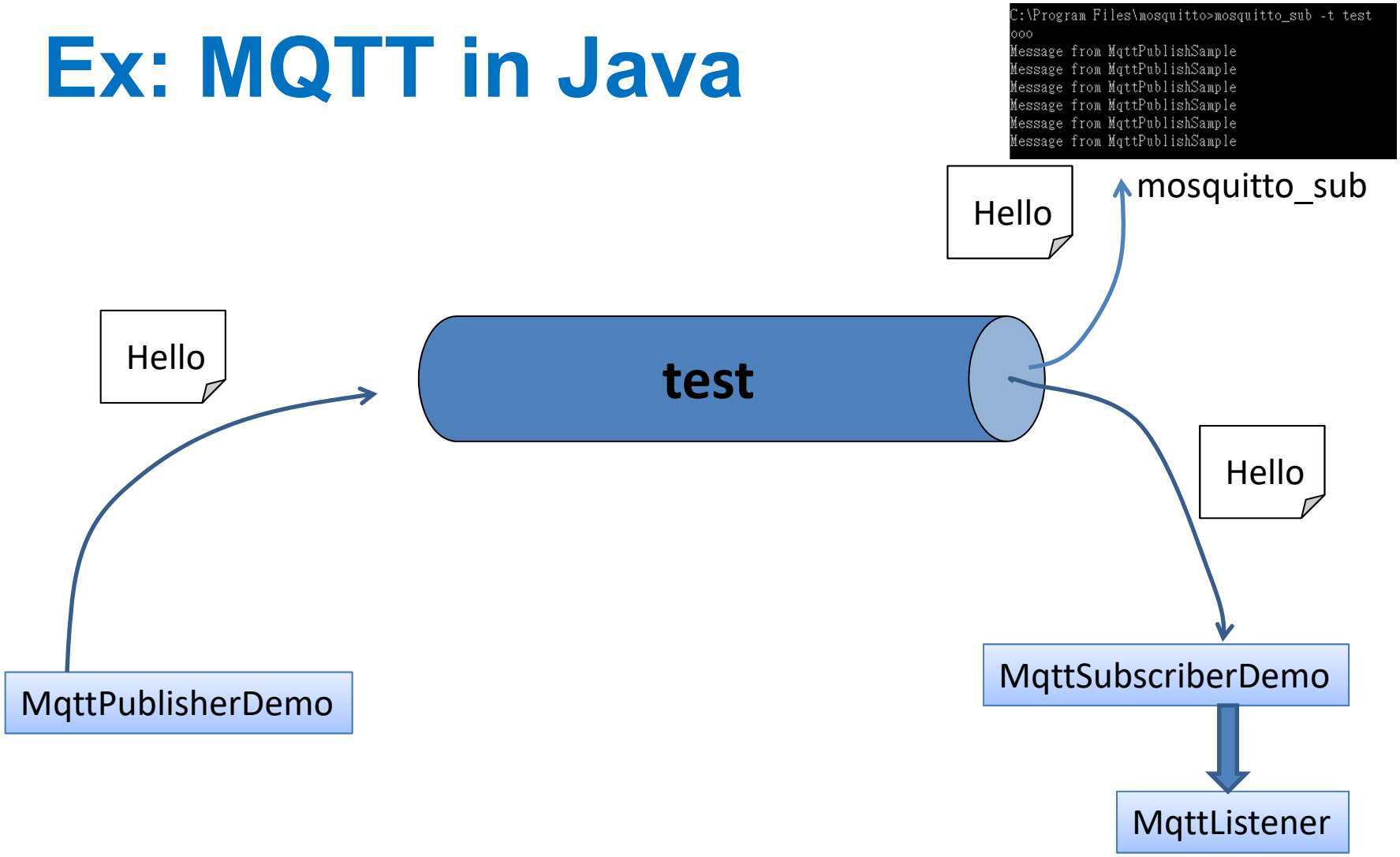


封包觀察

- See wireshark

封包解讀 https://github.com/mqttjs/mqtt-packet

Ex: MQTT in Java



Java MQTT Publisher

- Connect

```
String clientId = "cfliao-pub";
MemoryPersistence persistence = new MemoryPersistence();
MqttClient sampleClient = new MqttClient("tcp://localhost:1883", clientId, persistence);
sampleClient.connect(connOpts);
```

- Publish

```
String content = “My Message";
String topic = “test”;
MqttMessage message = new MqttMessage(content.getBytes());
message.setQos(0);
sampleClient.publish(topic, message);
```

Java MQTT Subscriber

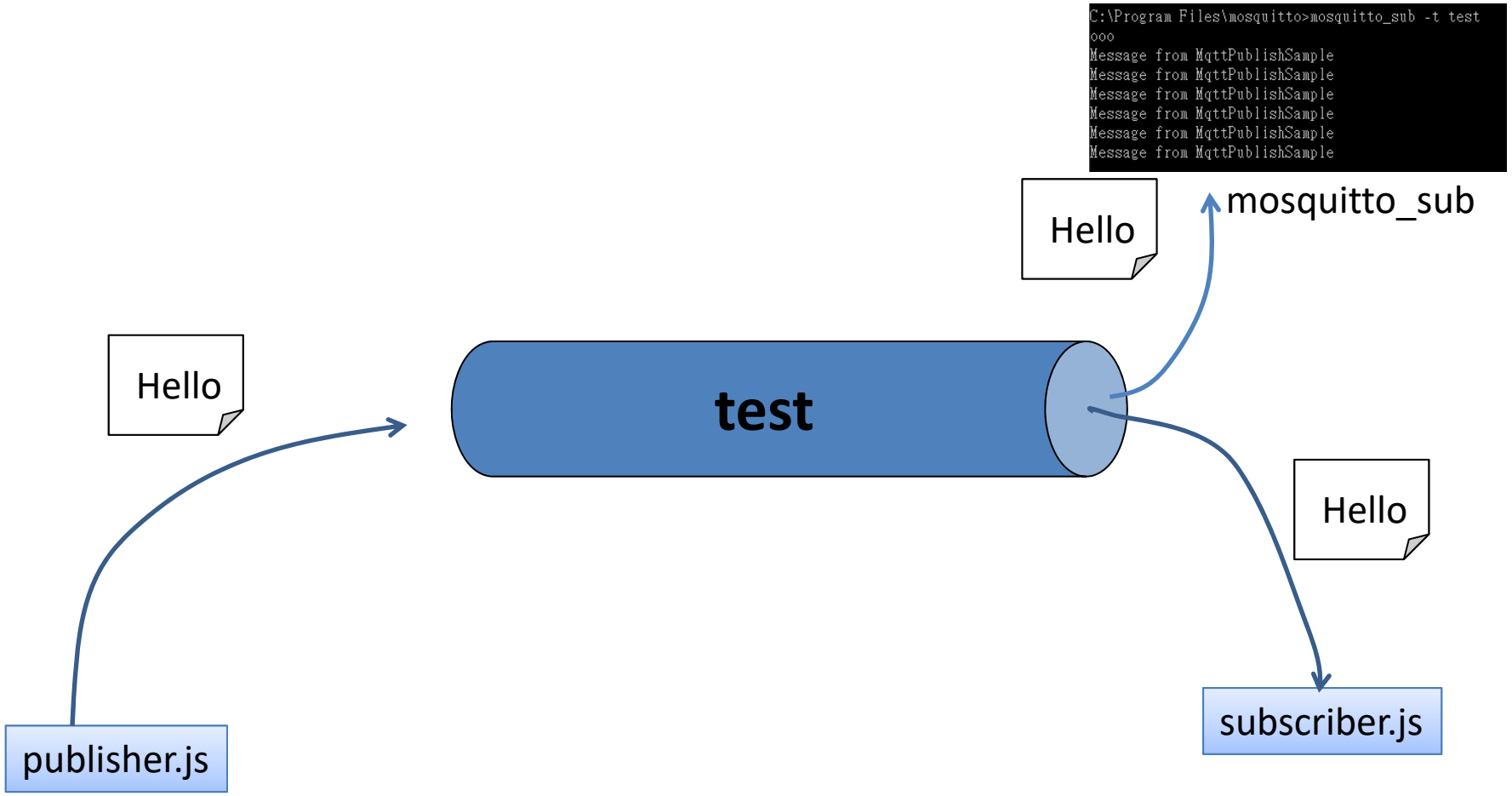
```
String clientId = "cfliao-sub";
MemoryPersistence persistence = new MemoryPersistence();
MqttAsyncClient sampleClient =
                    new MqttAsyncClient("tcp://localhost:1883", clientId, persistence);
sampleClient.setCallback(new MqttListener());
IMqttToken conToken = sampleClient.connect();
conToken.waitForCompletion();

String topicName = "test";
IMqttToken subToken = sampleClient.subscribe(topicName, 0);
subToken.waitForCompletion();
```

```
public class MqttListener implements MqttCallback {
        …
   @Override
   public void messageArrived(String topic, MqttMessage mqttMessage) throws Exception {
        System.out.println(mqttMessage.toString());
   }

}
```

Ex: MQTT in JavaScript



JavaScript (Node.js)

publisher

```
const mqtt = require('mqtt');
const client = mqtt.connect();
client.on('connect', function ()  {
   client.publish('MY_TOPIC', 'hello from js');
   client.end();
});
```

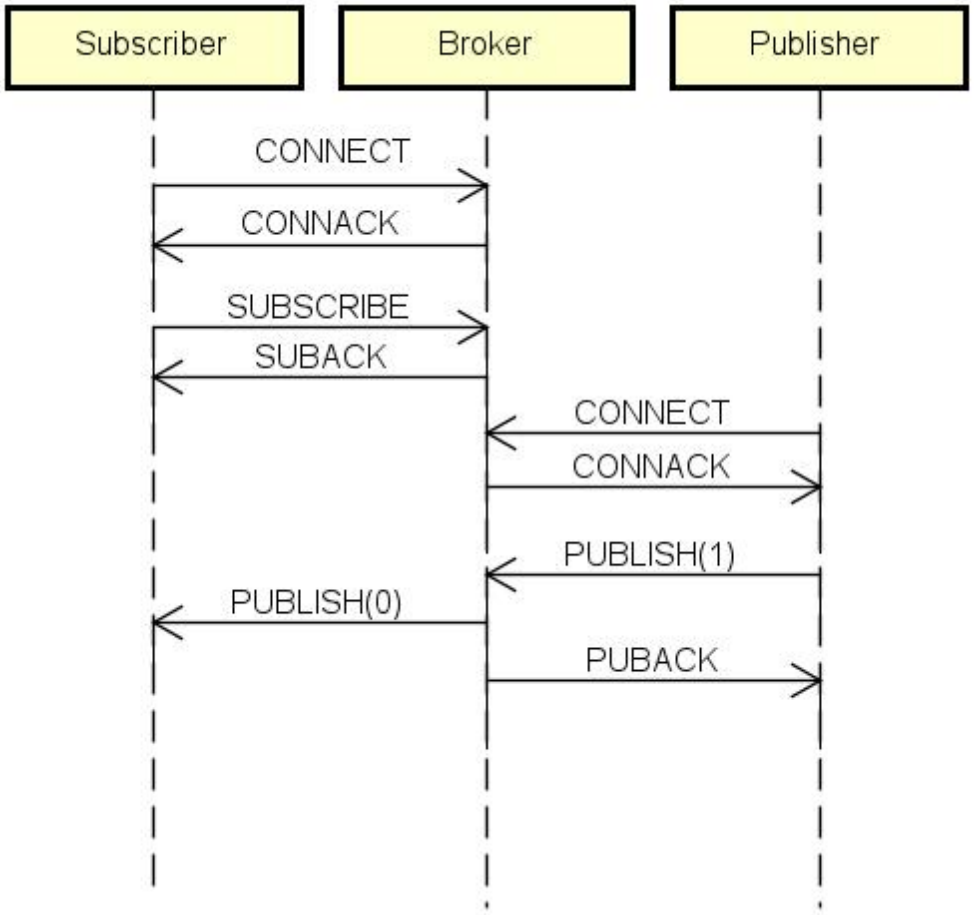
subscriber

```
const mqtt = require('mqtt');
const client = mqtt.connect();
client.subscribe('MY_TOPIC');
client.on('message', function (topic, message) {
   console.log(message.toString());
   client.end();
});
```

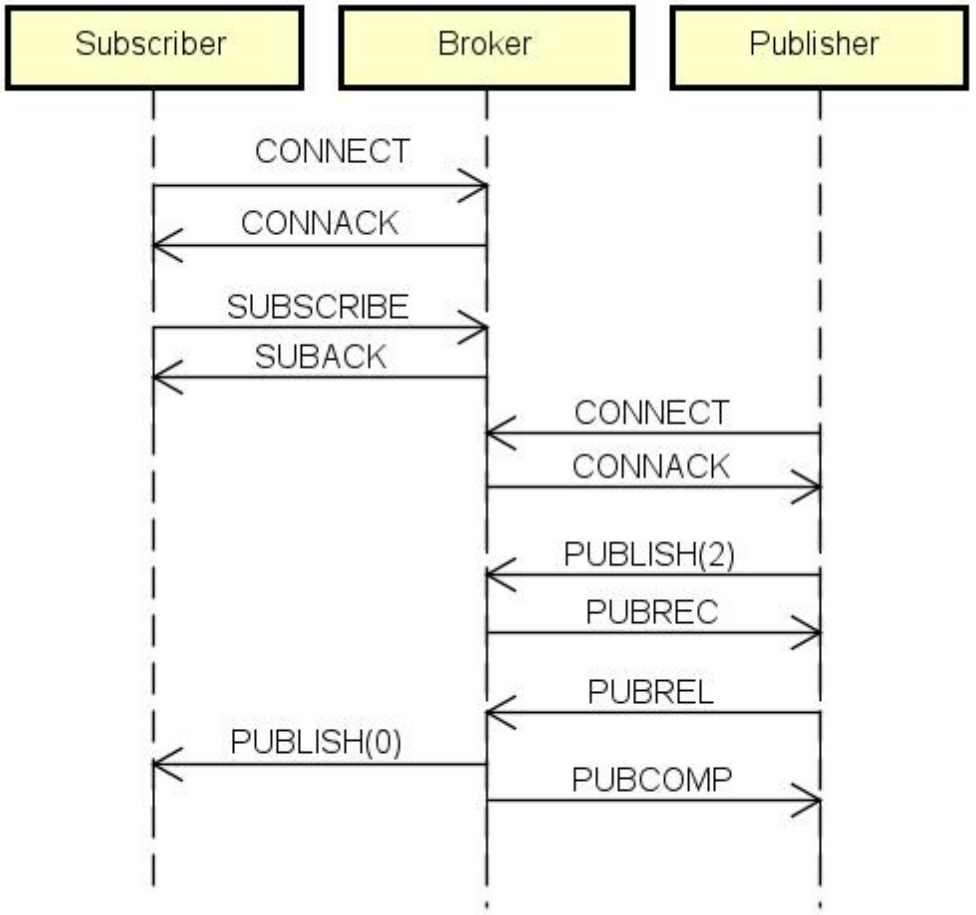
Case: MQTT QoS and Last Will

- QoS 0 “At most once”，最多一次
  - 訊息遺失或是重複發送的狀況可能會發生
  - Ex: 溫度感測
- QoS 1 "At least once"，至少一次
  - 採用ACK進行訊息確認送達，有問題則重送
  - 若遇到non-idempotent 邏輯，訊息重複會造成錯誤
- QoS 2 "Exactly once"，確定一次
  - 確認訊息只會送到一次
  - 耗費較多資源與網路頻寬
- Last Will
  - 連線後可事先設定Last Will，敘明當異常斷線發生時的處理方式

Case: MQTT QoS 1 (Publisher-side only)



Case: MQTT QoS 2 (Publisher-side only)

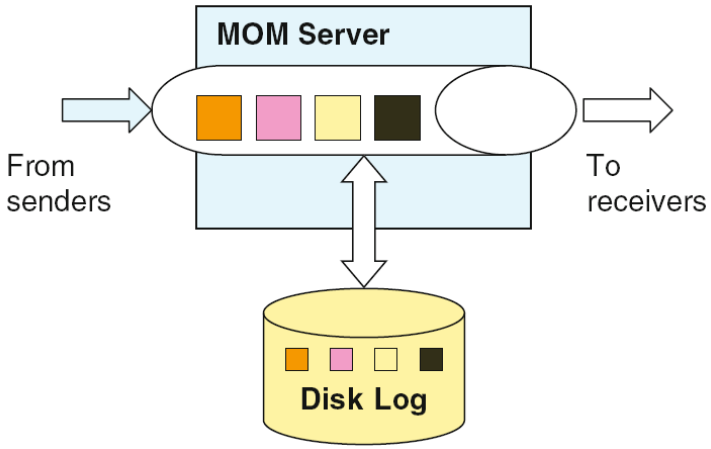


Reliable Message Delivery

- Reliability comes with the price of performance
- 三種常見的broker支援reliable message方式
  - Best effort
  - Persistent
  - Transactional

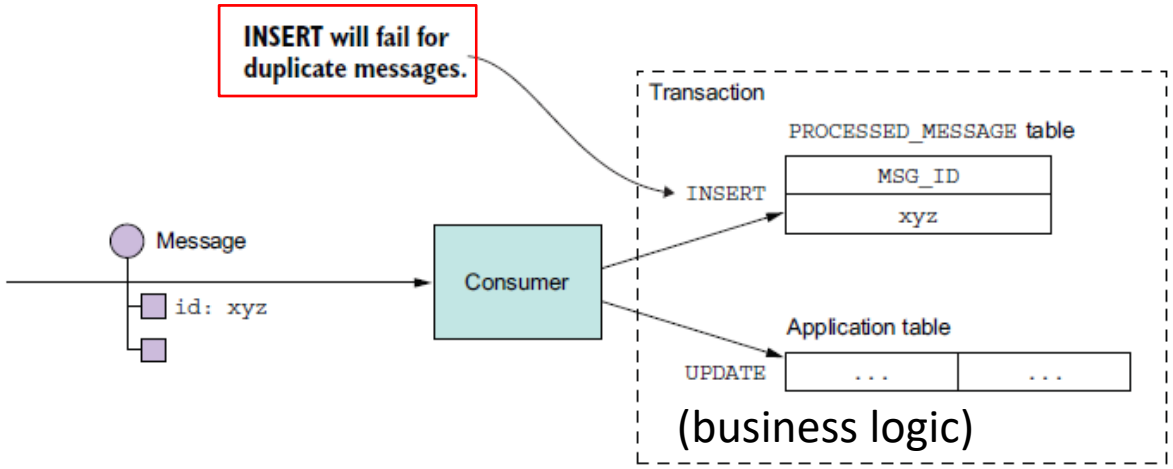
Reliable Message Delivery

- Best effort
  - Undelivered messages are only kept in memory and can be lost if a system fails
- Persistent
  - Undelivered messages are logged to disk
  - Can be recovered and delivered after a system failure



Handling duplicate messages

- Two options
  - Idempotent (重覆訊息不影響邏輯正確性) message handler
  - Track messages and discard duplicates
    - A simple implementation (using DB transaciton)
      1. Insert message_id of each message into a local database
      2. Message_id as a primary key: key重覆的話，此交易就失敗

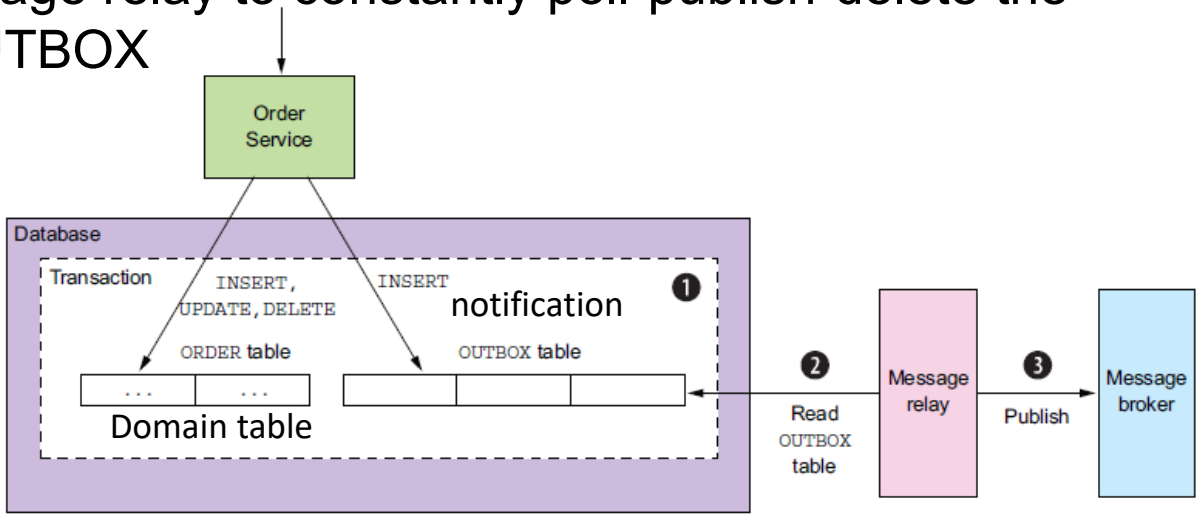


Transactional Messaging

- Motivation and problem
  - A message endpoint need to publish an event when it updates its states
    - Update → Publish
  - “Update without notification” can occur
    - Update → (endpoint crashes)

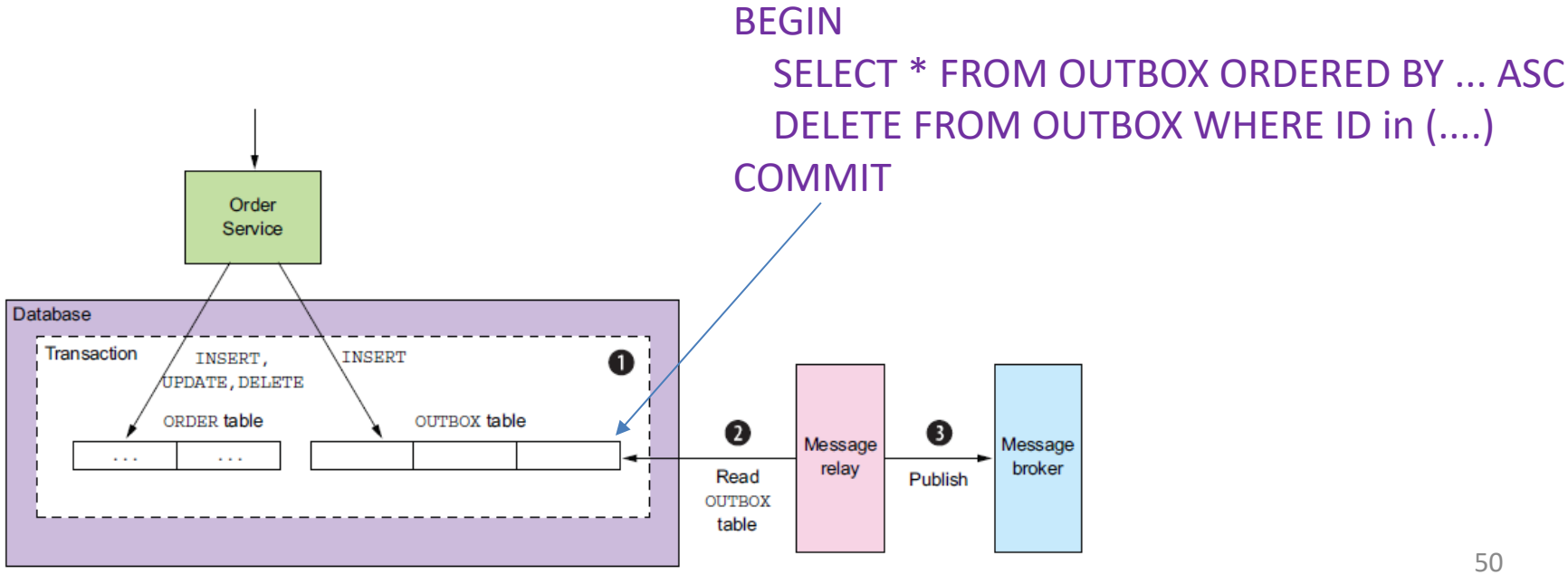
Transactional Outbox

- Solution: 確保state修改, notification一定會送出
  - Write the message to be published into a OUTBOX table
  - Updates to domain tables and the OUTBOX table are bundled into a Transaction
    - 確保「更改domain table」與「notification」一同被完成
      - 例: 完成ORDER修改→ service crash → OUTBOX還沒改所以rollback
  - Using a message relay to constantly poll-publish-delete the records in OUTBOX



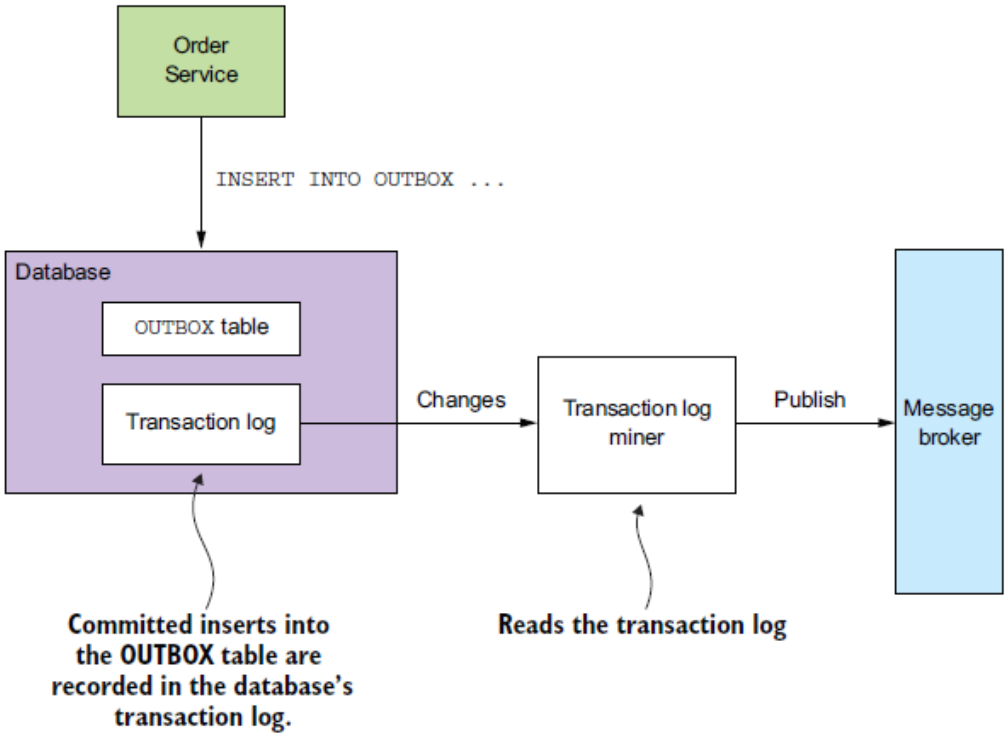
Realizing Message Relay

- Polling publisher (Message Relay)
  - The message relay constantly query the OUTBOX, publish events, and then clear OUTBOX in a transaction
  - Cons: Frequently polling the database can be expensive



Realizing Message Relay

- Transaction Log Tailing (又稱CDC, Change Data Capture)
  - Use a transaction log miner to read the low-level transaction logs of the DB and publish each change to the message broker

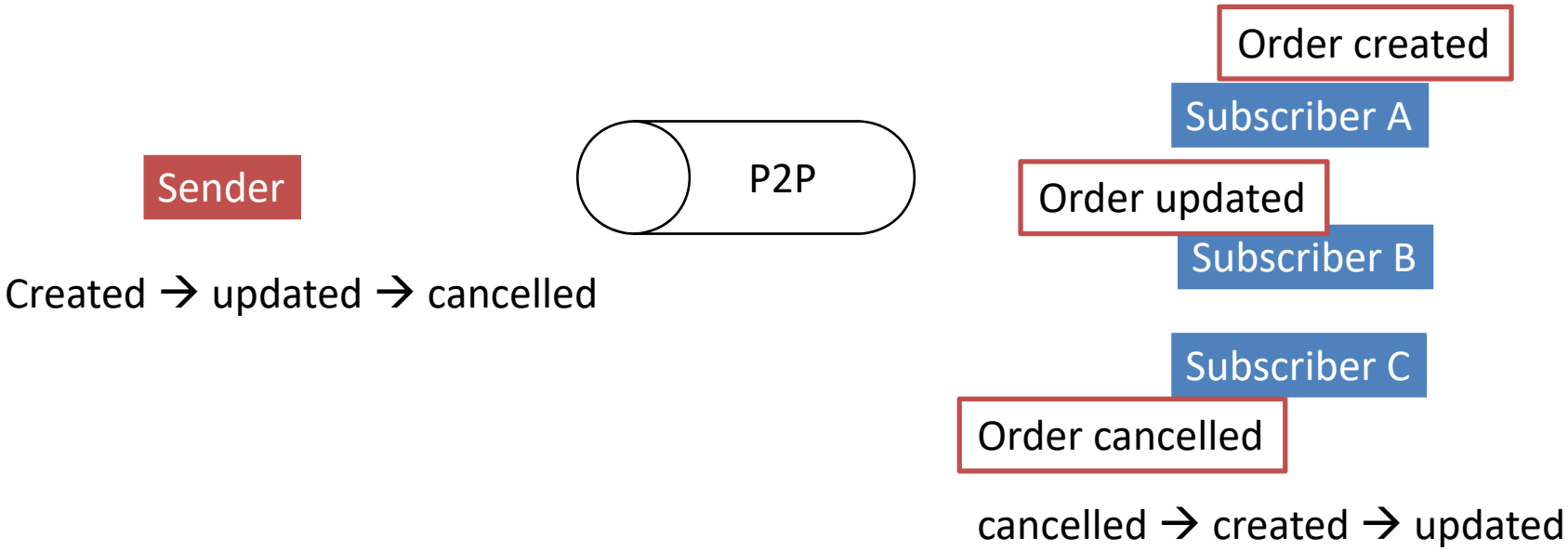


Realizing Message Relay

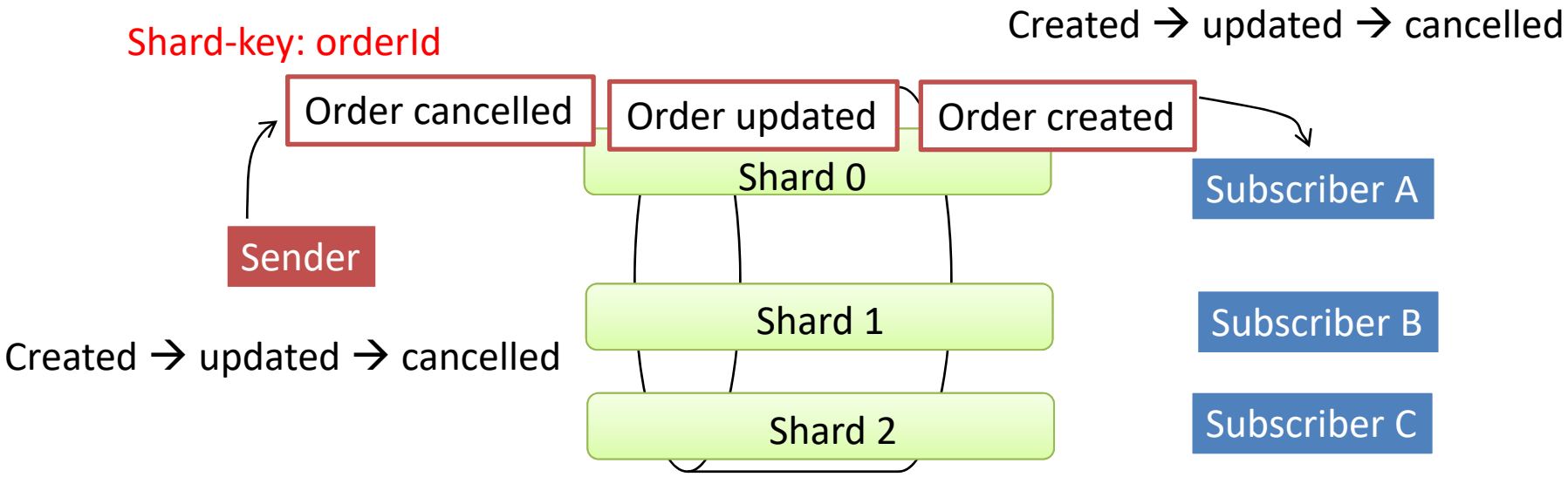
- Transaction Log Tailing Tools
  - Debezium
    - publishes database changes to the Apache Kafka message broker
  - Eventuate Tram (by Chris Richardson)
    - An enhanced version of Debezium
  - LinkedIn Databus
    - mines the Oracle transaction log
  - DynamoDB streams
    - DynamoDB built-in service

Message Ordering Problem

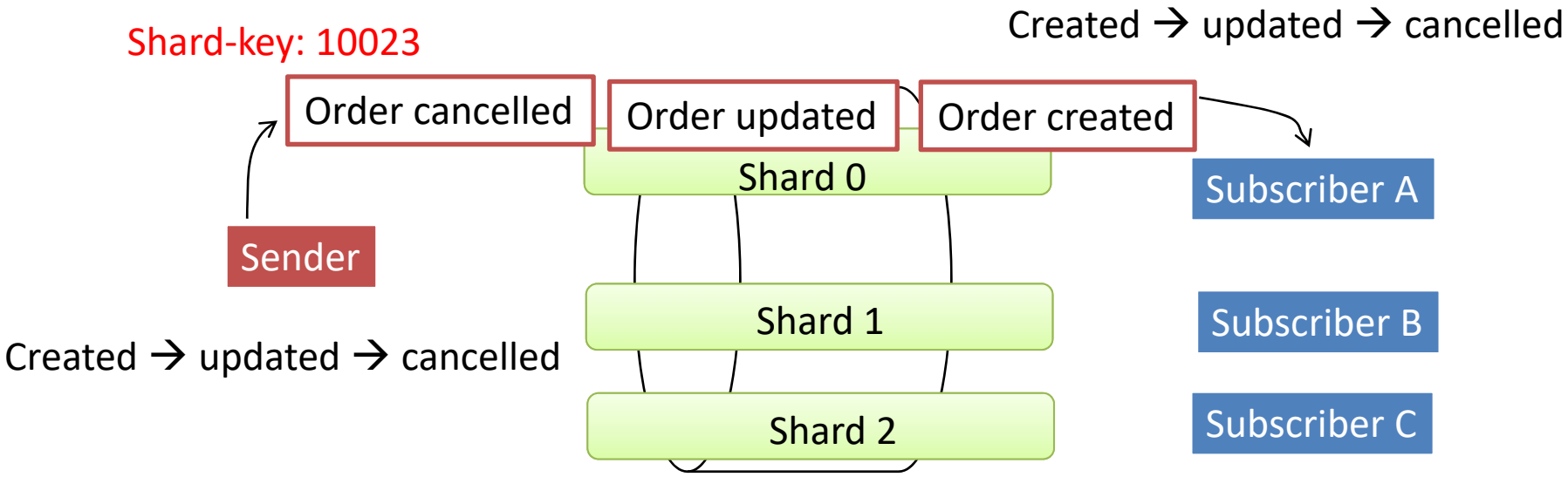
- Multiple subscribers use the same topic
  - 希望做load balance，但又希望某些訊息依次序接收
  - Example: (order created, order updated, order cancelled)



Channel Sharding

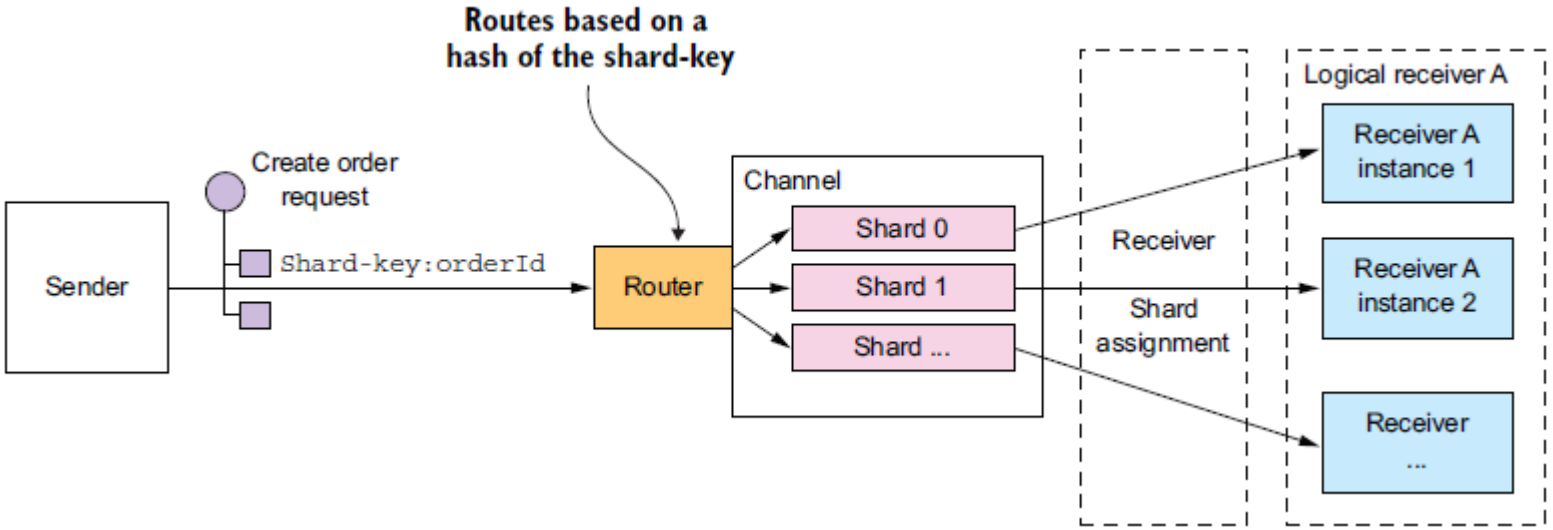


將需要具有次序的一群訊息，以shard-key group起來
MOM會將它們綁定同一個shard queue中; 被同一個subscriber處理


Shard-key: 10023
Created → updated → cancelled
Order cancelled
Order updated
Order created
Shard 0
Subscriber A
Sender
Shard 1
Subscriber B
Created → updated → cancelled
Shard 2
Subscriber C

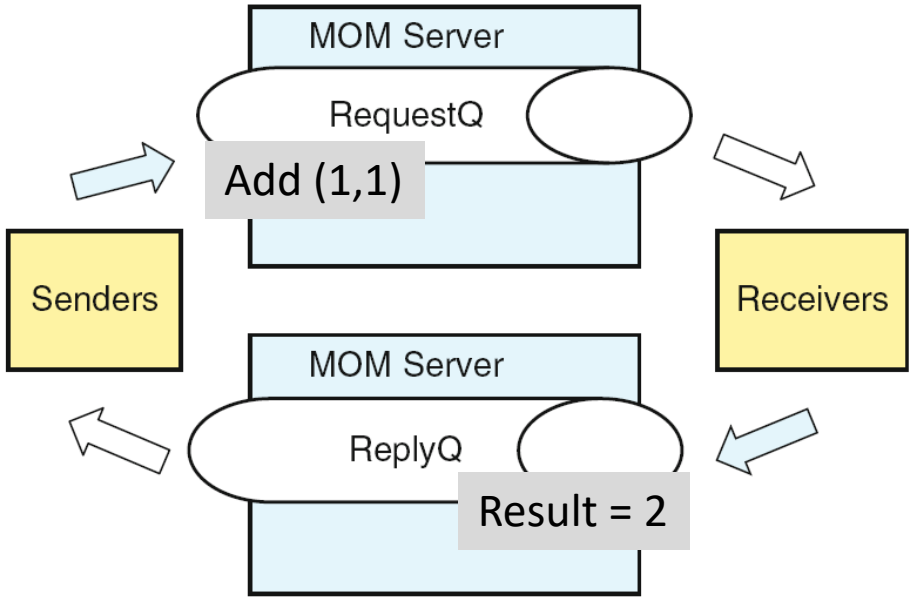
Channel Sharding

- Mechanism
  - A group of subscribers process the same channel
  - Broker splits a channel into two or more shards, each of which behaves like a channel
  - Sender specifies a shard key in header
  - Broker uses the shard key to assign the message to particular shard: 同一個shard key會送往同一個shard

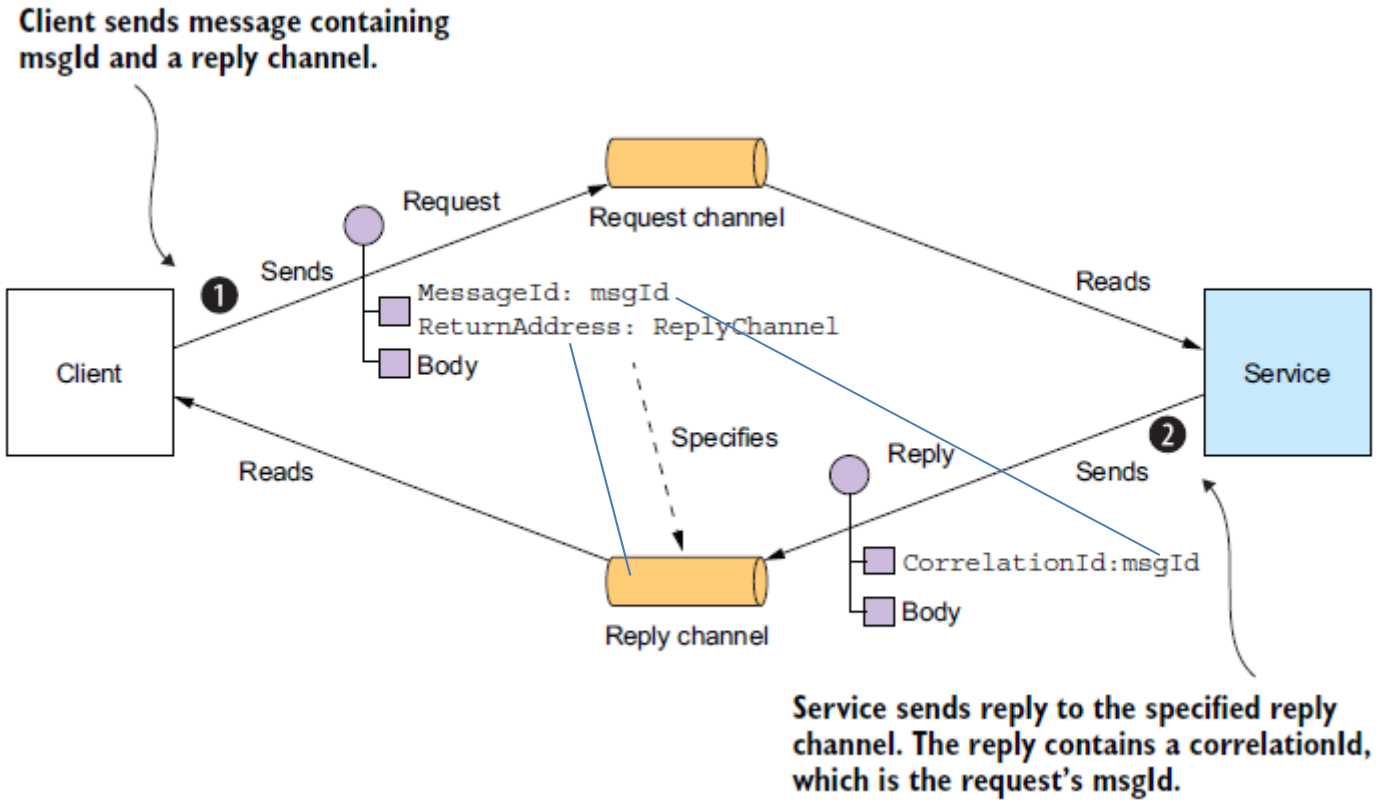


Simulating RPC

- MOM can also be used for synchronous communications
- Frequently used in enterprise systems to replace conventional synchronous technology

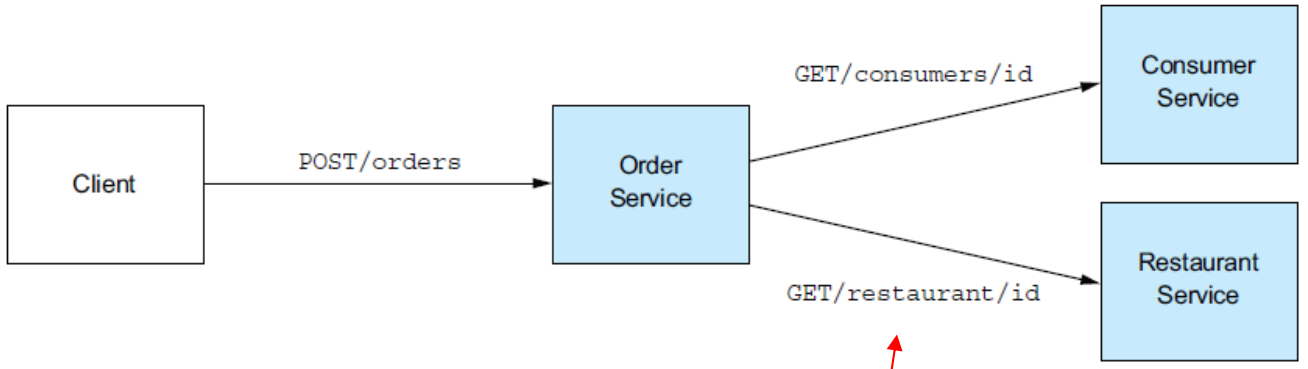


Simulating a Call



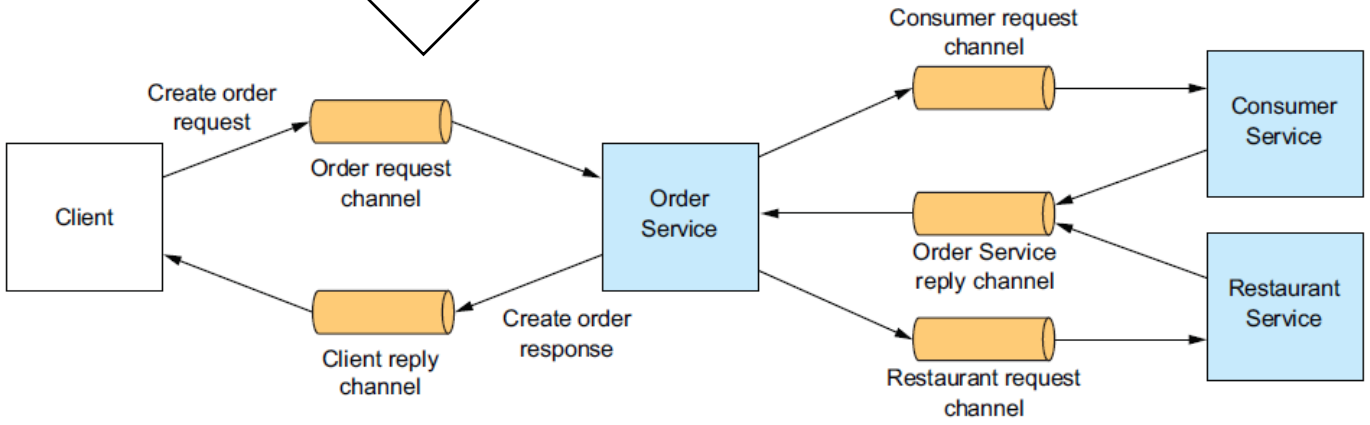
Case:提升RESTful WS的availability

- 如下圖中的同步RPC，呼叫時，三個服務必須同時available

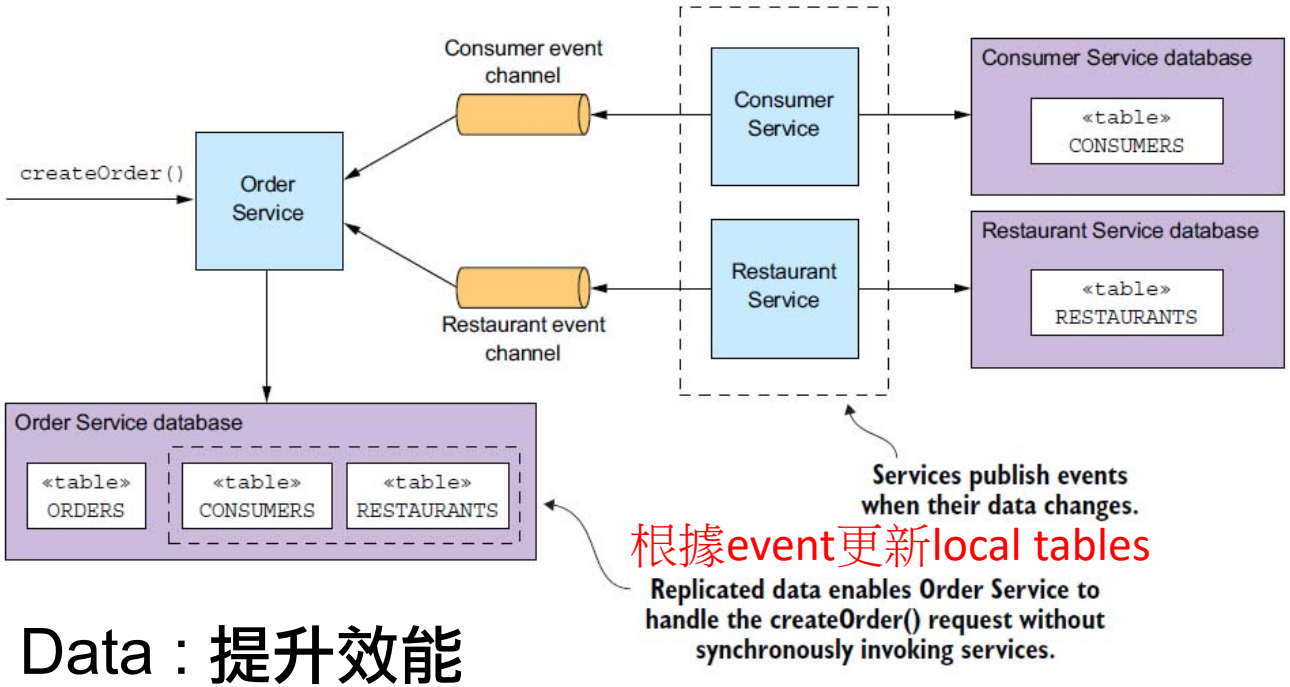


觀察: 只涉及讀取邏輯

變成非同步之後，若服務不available，訊息會先存在MQ中

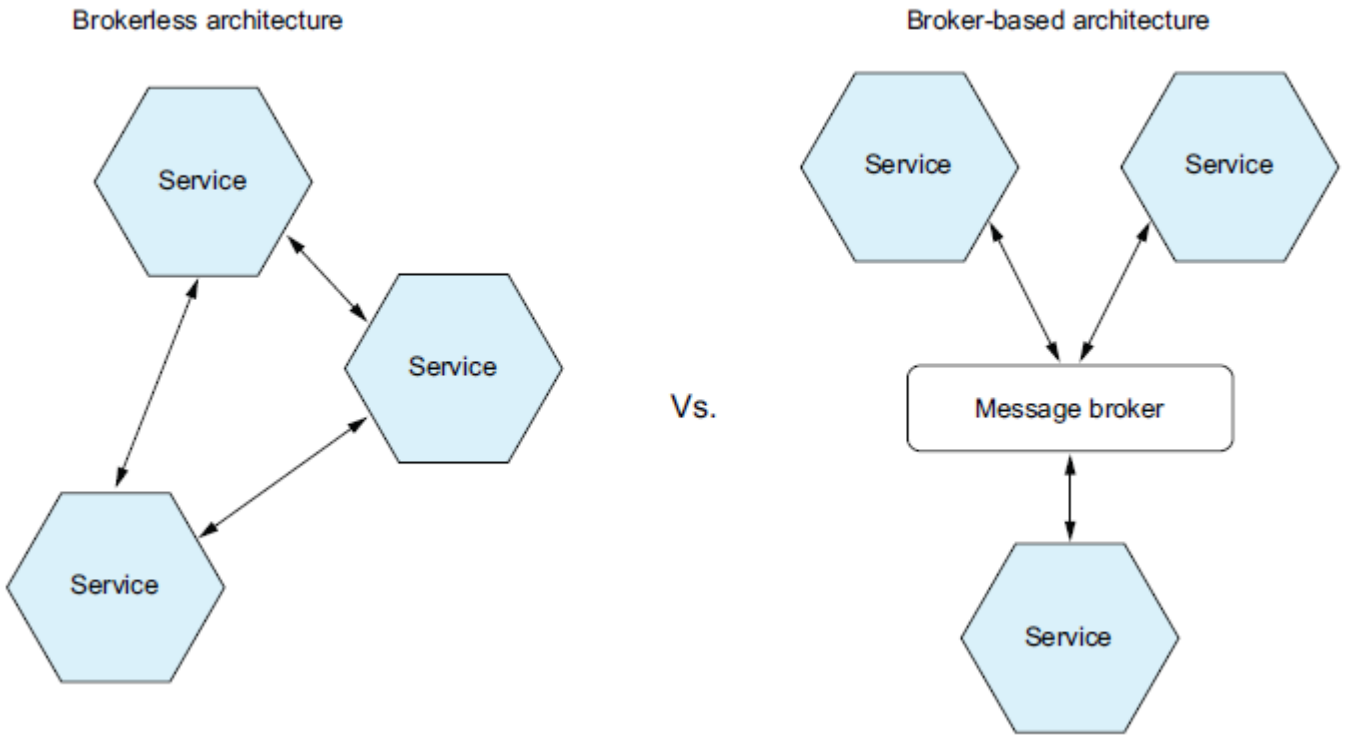


有更新時發出 event



- Replicate Data：**提升效能**
  - Order對Consumer與Restaurant都只涉及**查詢**
    - 在Order中維護一份Consumer與Restaurant的資料庫副本
    - Order中副本接收更新的Domain Event，有更新時才更新
    - 如此一來，原來2個查詢式RPC要求直接變成local db查詢
  - **缺點: 只適用於查詢**
    - Order Service無法直接對真正的table內容進行修改

Two Types of Message Brokers



In the broker-less architecture, each endpoint sees the “logical” address (or URI) of the virtual broker

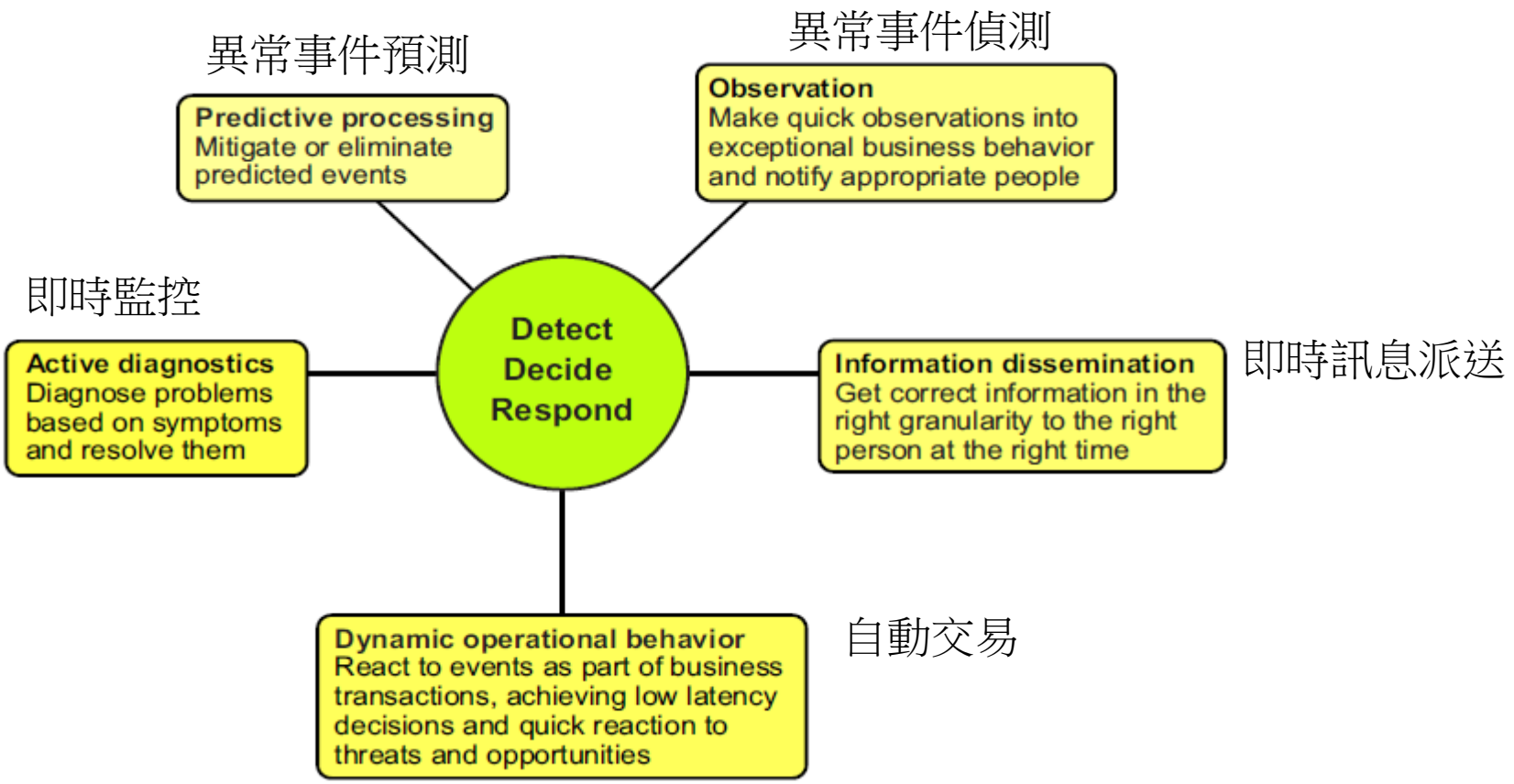
Brokerless Architecture

- Pros
  - Light network traffic and better latency
    - Messages go directly from the sender to the receiver
  - Prevent performance bottleneck or a single point of failure
  - Less operational complexity: no broker to setup and maintain
- Cons
  - Need to know about each other's locations
    - Use one of the discovery mechanisms
  - Offer reduced availability
    - Both the sender and receiver of a message must be available while the message is being exchanged (傳送過程中，沒有保存訊息之處)
  - Hard to implement guaranteed delivery
- Example
  - ZMQ、NSQ、multicast

Event as a Message (Domain Eventing)

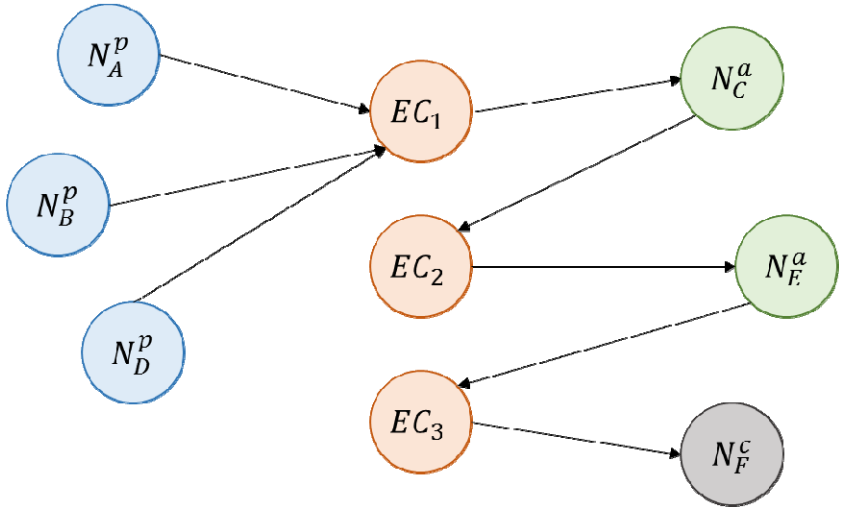
- Event
  - A digital indication of something has happened
  - Typically requires a reaction (computer processing)
- Suitable for
  - IoT systems
  - Robot systems
  - Interactive systems

EDA Applications



Channel-style Event Processing Network

- 由Sharon & Etzion提出，由四個部分組成
  - Event Producer(N^p): 只送不收
  - Event Consumer(N^c): 只收不送
  - Event Processing Agent(EPA) (N^a)
    - 收送皆有
    - 可對Event Producer 或其他EPA發出之事件做進一步處理
  - Event Channel(EC)



Complex Event

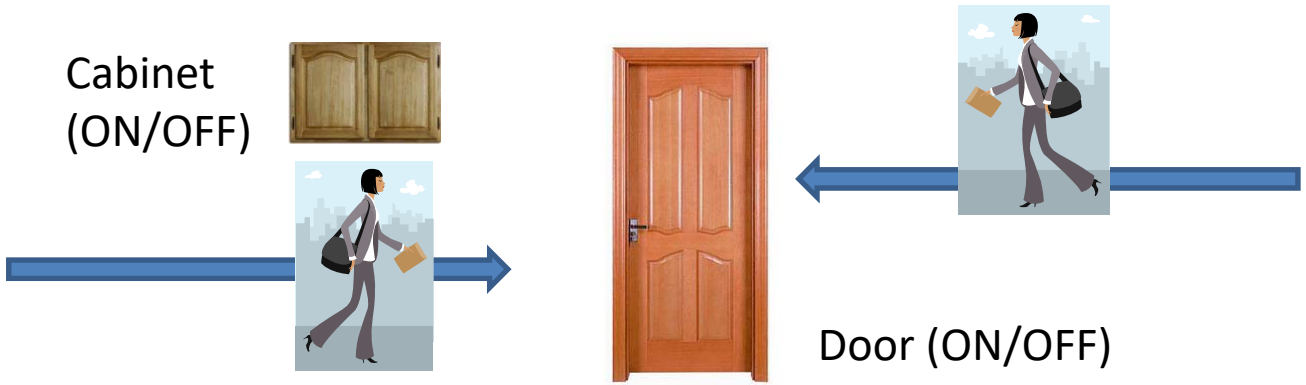
- An event is complex if $|C(e)| > 1$

$C(e) = (\varepsilon_1, \varepsilon_2, ...\varepsilon_n)$, where ε_i is the i th event that causes e to happen

e: event

C(leaveHome)=(Cabinet_ON, Cabinet_OFF, Door_ON, Door_OFF)

C(comeHome)=(Door_ON, Door_OFF, Cabinet_ON, Cabinet_OFF)

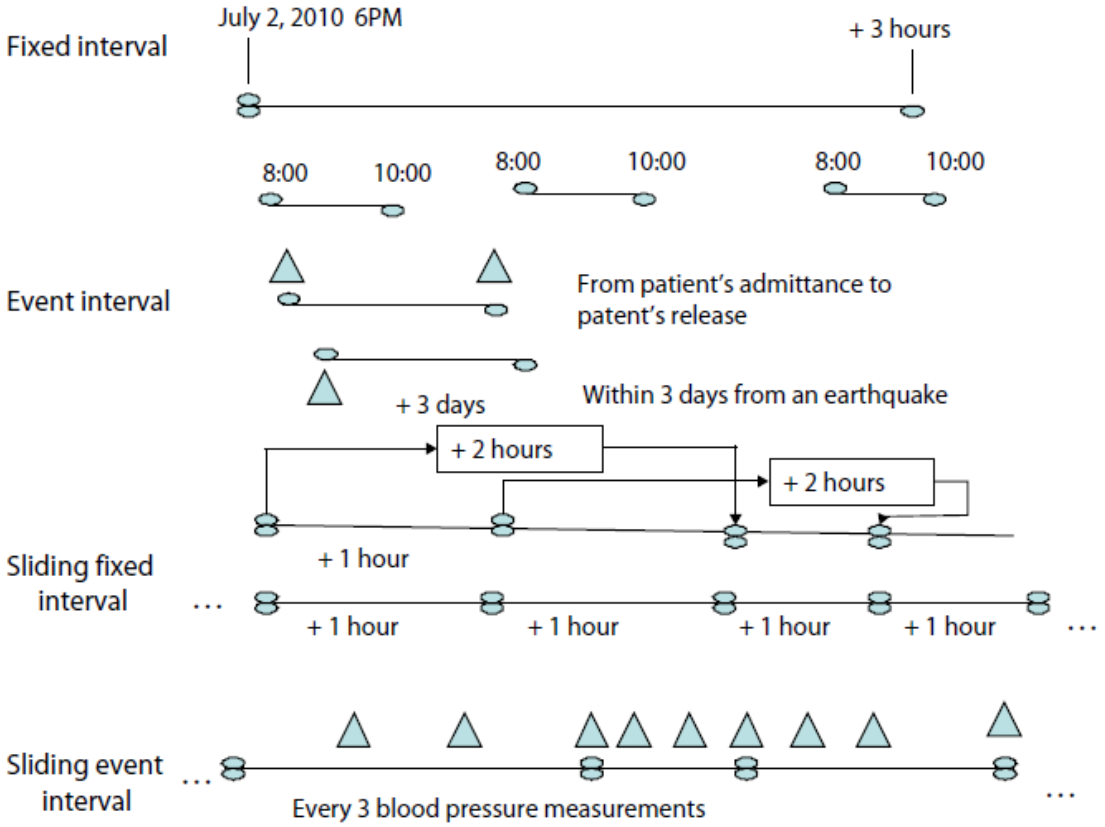


事件前提

- 事件前提(Context)
  - 若不符合此前提則EPA將不予處理
  - 透過事件處理前提的過濾機制，EPA可取得並處理特定事件

Temporal context

- 由一個或多個時間間隔組合成，有些會重疊發生



以絕對時間區分

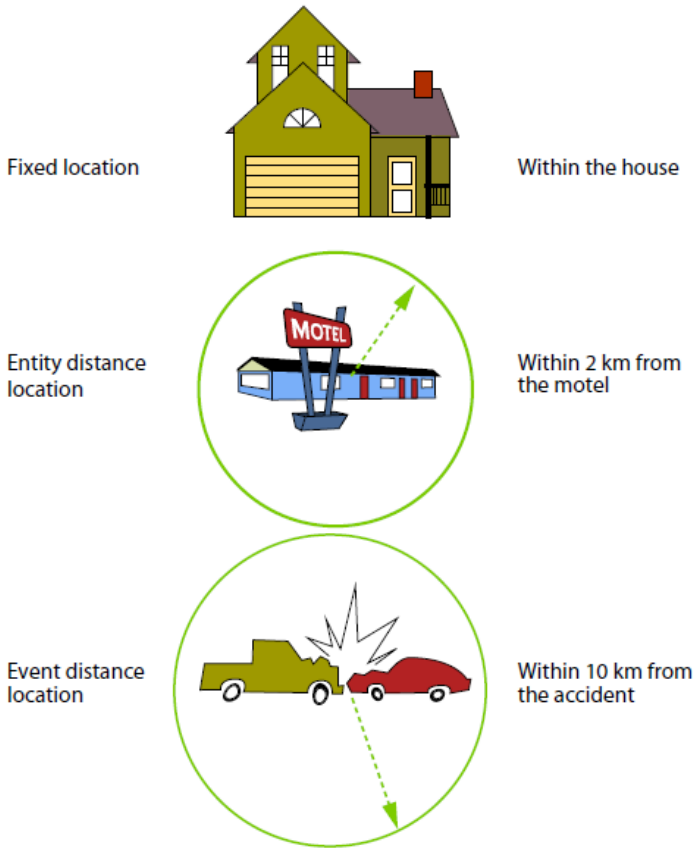
以事件發生區分

以相對時間區分

以事件發生次數區分

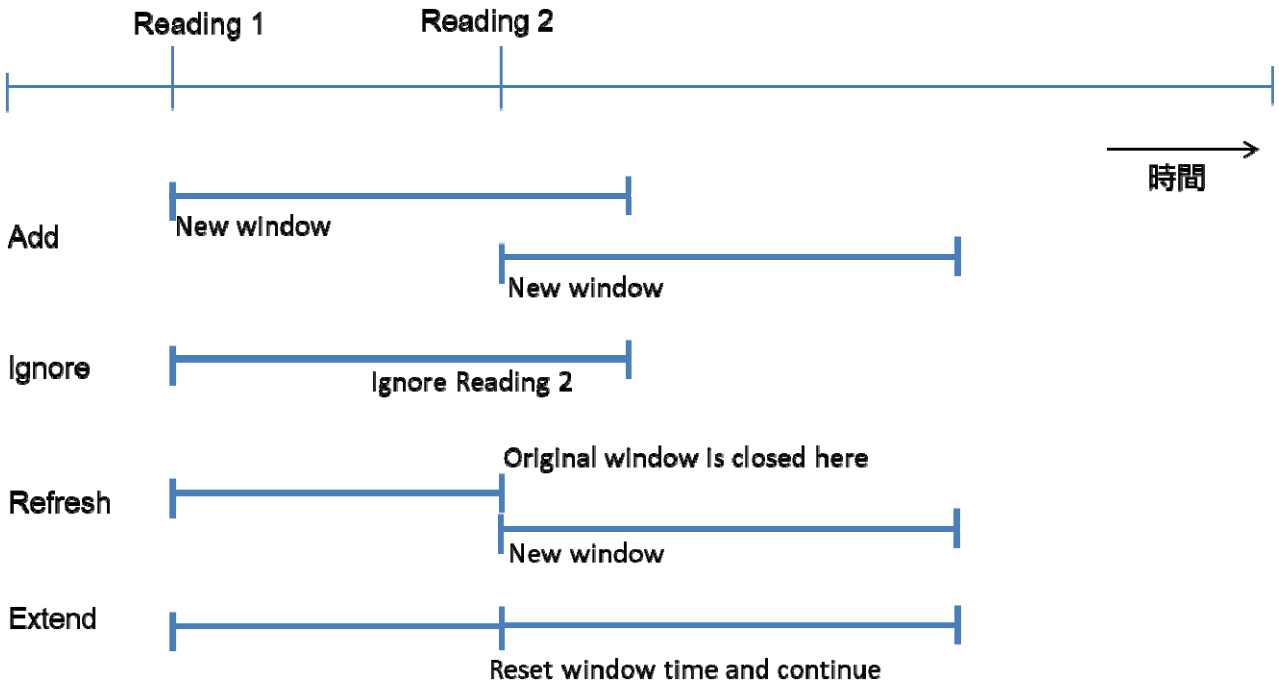
Spatial Context

- The location attribute can take two forms:
  - 座標(經緯度)
  - 空間位置名稱(大仁樓)
- Examples of spatial context:
  - 固定地點(Fixed location)
  - 實體距離 (Entity distance location)
  - 事件距離 (Event distance location)



Context Initiator Policy

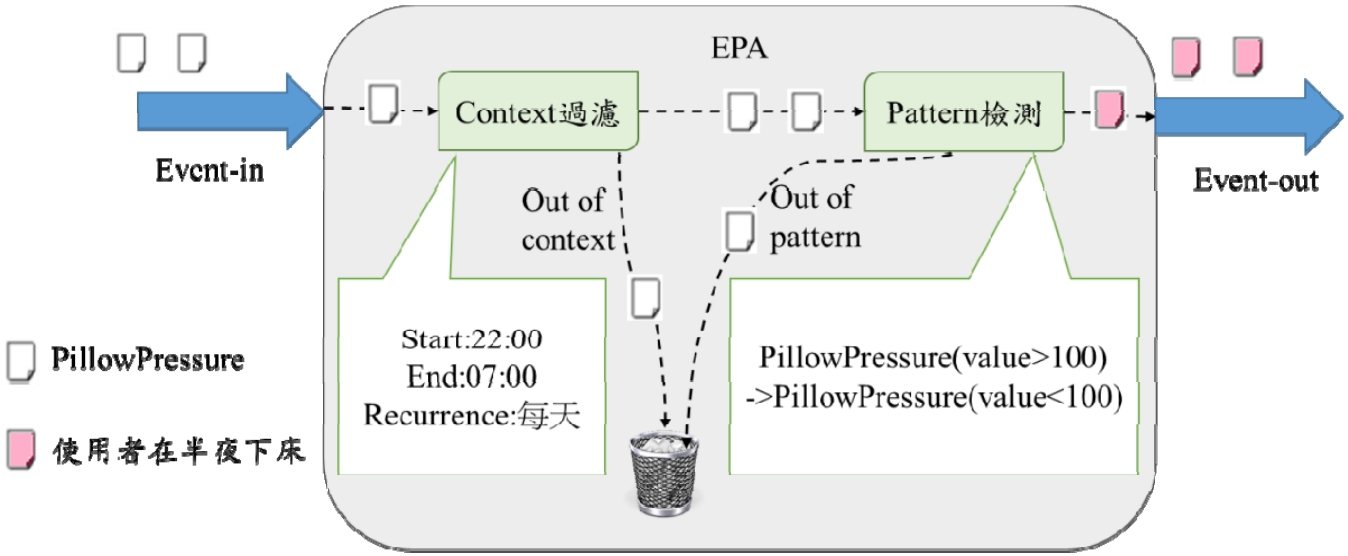
連續二個相同類型event進來時該如何處理?



例如，室溫大於28度→很熱所以開冷氣1小時，如果10秒間取得二個讀值28度以上，如何解讀?
Add: 開二次
Ignore: 只有第一次會開
Refresh: 先關再開
Extend: 從第二次到時，原來計算的時限(1小時)重設

範例1

- 當使用者在晚上10:00至早上7:00時下床，則開啟房間與浴室燈
  - 定義Context為重複的固定時間間隔
    - CREATE CONTEXT NightContext start 22:00 end 07:00
  - 定義Pattern為床的壓力感測器數據由>100轉變至<100
    - PillowPressure(value>100) -> PillowPressure(value<100)



範例2

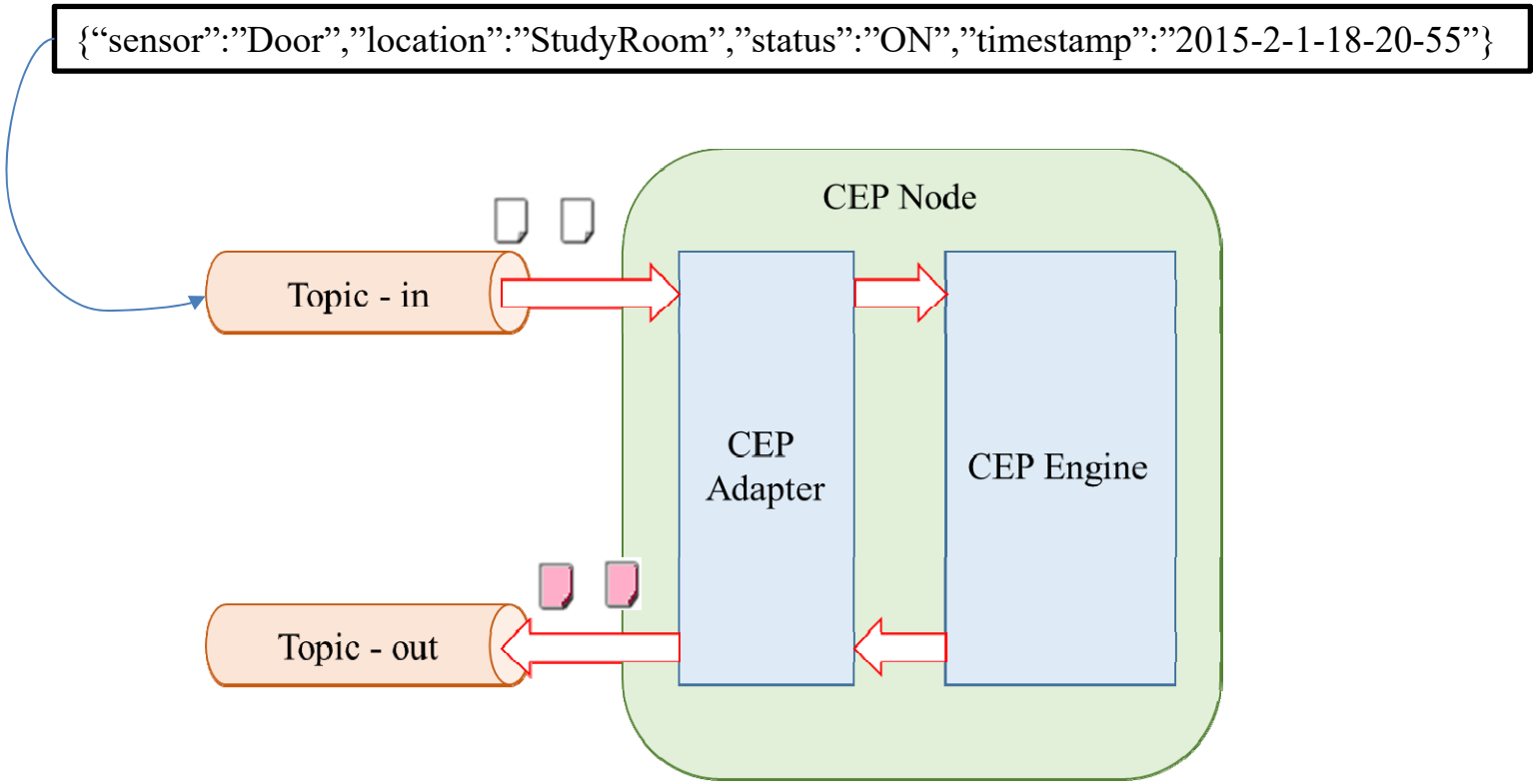
- 當使用者在晚上10:00至早上7:00時進入浴室後20分鐘未回到床上時，發出警告訊息
  - 定義Context為重複的固定時間間隔
    - CREATE CONTEXT NightContext start 22:00 end 07:00
  - 定義Pattern為浴室門感測器狀態由ON轉變至OFF，並且20分鐘內未發生床的壓力感測器數據>100
    - BathroomDoor(status = ‘ON’) -> BathroomDoor(status=’OFF’)

    ->(timer:interval(20 min) and not PillowPressure(value>100))

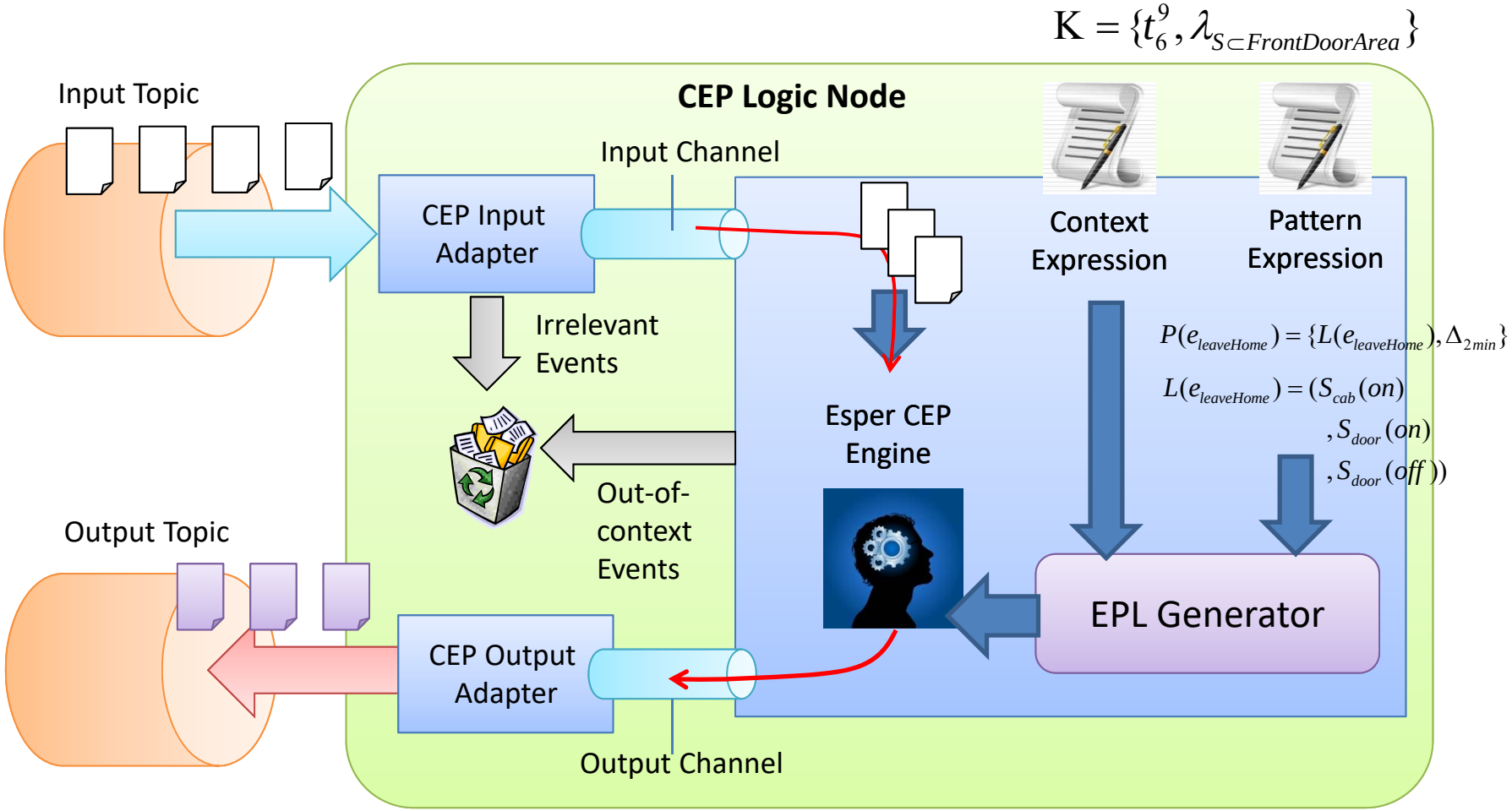
去廁所超過20分鐘還沒回來

CEP+MOM

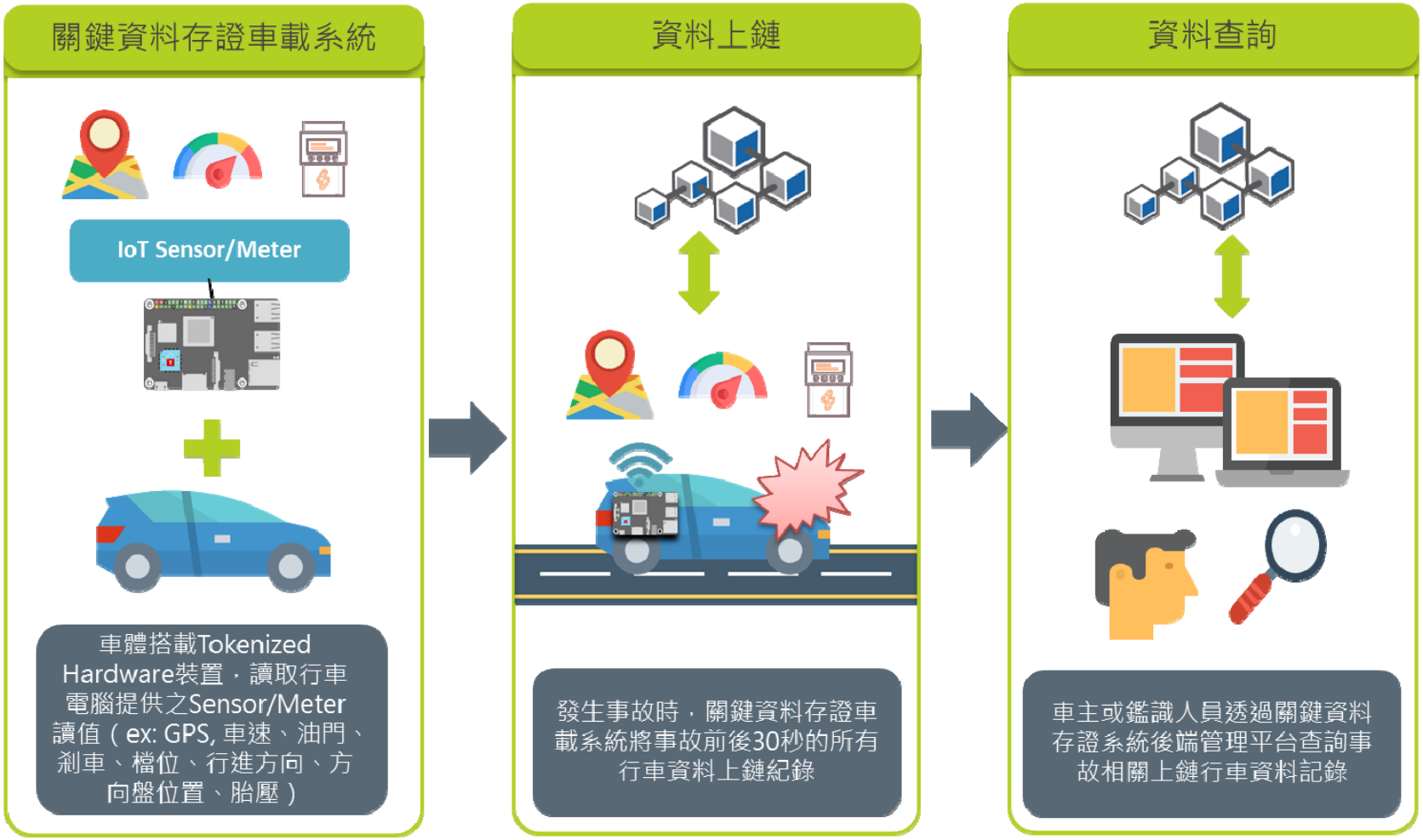
- 感測器數據透過CEP Adapter轉換成CEP引擎所接受之事件格式



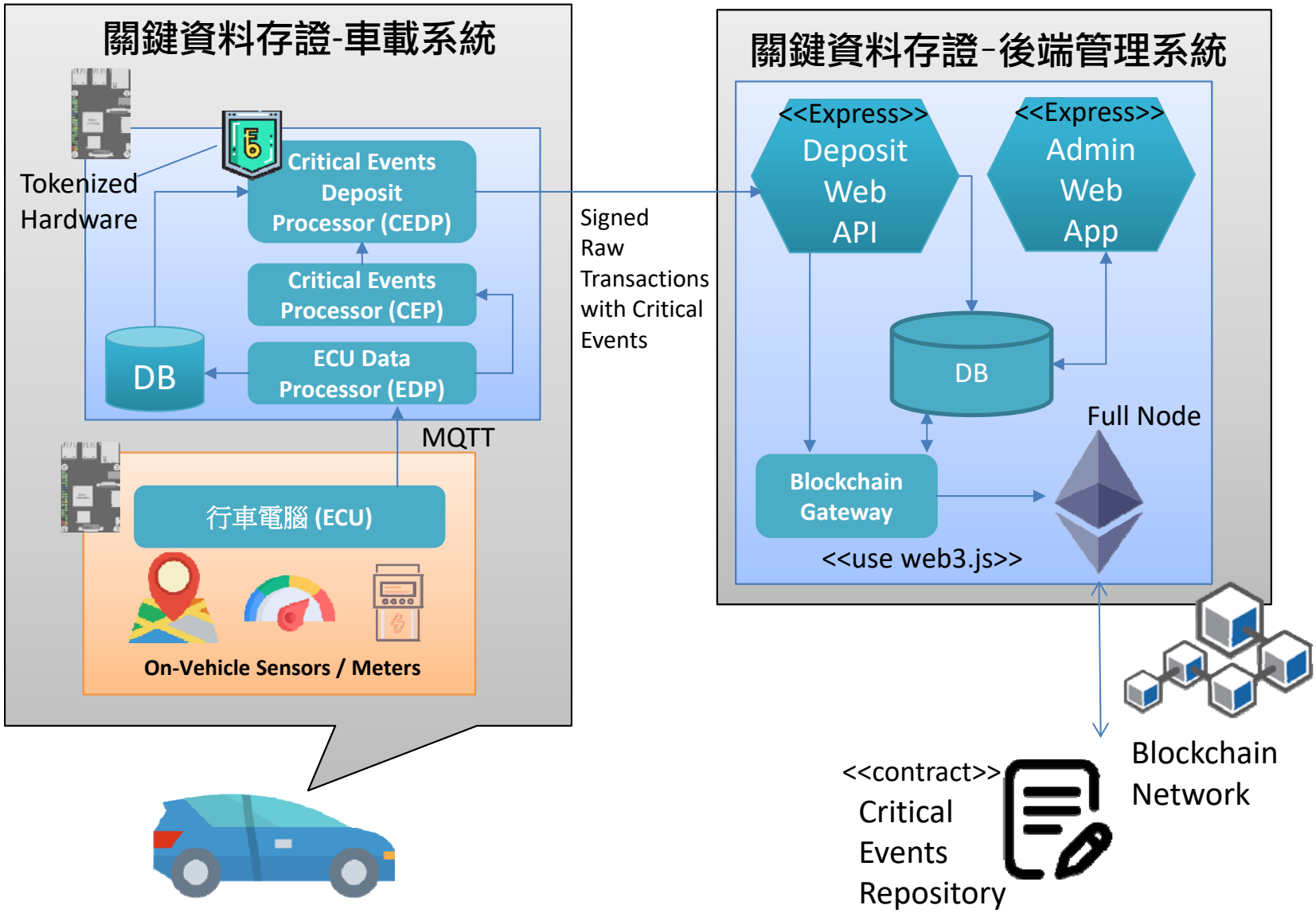
Architecture of a CEP + MOM



案例: 事故存證



工研院，基於區塊鏈技術的車聯網關鍵資料存證服務規劃與開發 (2021)


關鍵資料存證-車載系統
Tokenized Hardware
Critical Events Deposit Processor (CEDP)
Critical Events Processor (CEP)
ECU Data Processor (EDP)
DB
MQTT
行車電腦 (ECU)
On-Vehicle Sensors / Meters
Signed Raw Transactions with Critical Events
關鍵資料存證-後端管理系統
<<Express>> Deposit Web API
<<Express>> Admin Web App
DB
Full Node
Blockchain Gateway
<<use web3.js>>
Blockchain Network
<<contract>> Critical Events Repository

Lab: MQTT+Naïve CEP